Sistemas Informáticos
Curso 2005-2006

SIOS-m: Desarrollo y simulación de un Sistema Inteligente Orientado a Servicios Móviles.

Susana Bautista Blasco
Daniel González Polo
Elena Pilar Rueda Moltó

Dirigido por:
Prof. Ana M. González de Miguel.
Dpto. Sistemas Informáticos y Programación.

Facultad de Informática.
Universidad Complutense de Madrid.

PROYECTO DE SISTEMAS INFORMÁTICOS

SIOS-m: Desarrollo y Simulación de Un Sistema Inteligente Orientado a Servicios Móviles

By Susana Bautista Blasco, Daniel González Polo y Elena Pilar Rueda Moltó

Profesor Director: Ana M. González de Miguel

Resumen

El proyecto SIOS-m ha consistido principalmente en la creación de una plataforma de simulación orientada a servicios móviles. Para el desarrollo de la plataforma más idónea ha sido necesaria una investigación previa de las tecnologías propias de estos servicios. Y, partiendo de esta base, construir una aplicación inteligente que sea capaz de centralizar el proceso de comunicación entre el usuario final y los servicios ofrecidos por el proveedor, adaptándose en cada caso a las necesidades del usuario.

Abstract

The SIOS-m project has consisted primarily on the creation of a simulation platform oriented to mobile services. In order to develop the ideal platform for the project it has been necessary to perform a prior study of the technologies associated to these services and based on these studies, build an intelligent application. This application is capable of centralizing the communication process between the end-user and the services offered by the service provider, and furthermore adapt itself to the needs of each user.

Tabla de Contenidos

Lista de Figuras

Lista de Tablas

1. Introducción

1.1. Objetivos del Proyecto

El objetivo principal del proyecto SIOS-m será desarrollar y simular un *Sistema Inteligente Orientado a Servicios Móviles* (SIOS-m) utilizando conocimientos en técnicas de *Inteligencia Artificial* (IA) y tecnologías avanzadas en el dominio de la telefonía móvil.

Las plataformas de servicios móviles necesitan estar dotadas de un tipo de inteligencia artificial capaz de tratar situaciones imprevistas y tomar decisiones sin la intervención del papel humano. Por lo que necesitaremos dotar al sistema, de la inteligencia capaz de controlar situaciones nuevas y aprender de ellas.

1.2. Actividades del Proyecto

El proyecto SIOS-m consta de dos fases: una primera fase de adquisición de conocimientos y modelado previo de los servicios móviles, incluyendo un estudio detallado sobre el estado actual de las plataformas, arquitecturas, tecnologías, etc., y una segunda fase de análisis y diseño de todos los agentes que forman parte del entorno genérico de operación, así como de la plataforma de simulación.

A continuación detallamos las actividades de las dos fases del proyecto.

Fase I: Adquisición de Conocimientos y Modelado Previo de los Servicios Móviles

- Investigación del estado actual de las plataformas de servicios móviles (arquitecturas, tipos de servicios y tecnologías y/o estándares).
- Introducción a las Técnicas de Inteligencia Artificial.
- Estudio y selección de arquitectura, servicios móviles, técnicas IA y tecnologías adecuadas para la implantación del sistema SIOS-m.
- Modelación del entorno genérico de operación en términos de tipos de usuarios, dispositivos móviles, tipos de servicios y módulos funcionales de registro y entrega de servicios.
- Diseño de taxonomía de servicios móviles en el entorno de operación modelado.
- Estudio de Viabilidad de Sistemas Inteligentes.
- Análisis funcional y diseño de los módulos software de la Arquitectura SIOS-m.
- Diseño de un prototipo y plataforma de simulación (herramienta) adecuada para la Arquitectura SIOS-m.
- Definición de un Plan de Desarrollo.

Fase II: Desarrollo y Simulación de un Prototipo SIOS-m

- Elaboración de un Plan de Desarrollo y Simulación de un Prototipo SIOS-m basado en la descripción y planificación detallada de las siguientes tareas:
    a. Análisis, diseño e implementación de un Prototipo de la Arquitectura SIOS-m.
    b. Instalación, configuración y prueba de la Plataforma de Simulación.
    c. Definición de un Plan de Pruebas Funcionales y de Rendimiento del Prototipo SIOS-m.
    d. Instalación y configuración del Prototipo SIOS-m.

e. Realización del Plan de Pruebas Funcionales y de Rendimiento del Prototipo SIOS-m.

f. Análisis de Resultados y Optimización de la Arquitectura SIOS-m.

- Ejecución del Plan de Desarrollo y Simulación del Prototipo SIOS-m, documentando todos los resultados obtenidos.
- Establecer al menos tres líneas de evolución del trabajo realizado, incluyendo posibles adaptaciones de la Arquitectura SIOS-m a diversos entornos de operación.

2. Proyecto SIOS-m

2.1. Fase I: Adquisición de Conocimientos y Modelado de Servicios Móviles

Tras haber realizado diversas investigaciones en el proyecto, documentándonos tanto en las reuniones de trabajo llevadas a cabo semanalmente durante la primera fase con la Profesora Tutora del proyecto, como en consultas de libros, búsquedas en internet y el conocimiento adquirido a lo largo de la carrera, detallamos a continuación cada fase de investigación.

2.1.1. Investigación del Estado Actual de las Plataformas de Servicios Móviles

Revisando el estado actual de las plataformas móviles, hemos investigado los siguientes aspectos relacionados con el diseño de estas arquitecturas y las últimas tecnologías utilizadas:

- Arquitecturas Móviles (para más información ver, por ejemplo, [1], [2], [3], [4]).
- Tipos de Servicios (para más información ver, por ejemplo, [15],[16]).
- Tecnologías y Estándares Móviles [5][6][7].

2.1.1.1. Arquitecturas Móviles

En los últimos años la mayoría de procesos de negocio han cambiado en flexibilidad, interconectividad y autonomía debido a las condiciones del mercado, a los nuevos modelos organizacionales y a los escenarios de uso de los sistemas de información. En este contexto, internet está cambiando la forma en la que se ofrecen los negocios y los servicios a la sociedad global, y en la que estos negocios interoperan. Esta tendencia nos lleva a sistemas de información conectados e integrados a través de la infraestructura que proporciona internet. Internet introduce un nuevo entorno donde el software se va a ofrecer y acceder como servicio. Los Web Services proporcionan la plataforma tecnológica ideal para conseguir la completa integración de los procesos de negocio de una organización con diferentes organizaciones. Los Web

Services prometen ser el mecanismo adecuado para la implementación de las Arquitecturas Orientadas a Servicios (SOA) para sistemas de información integrados y distribuidos.

a) Arquitectura Orientada a Servicios (SOA)

La Arquitectura Orientada a Servicios (*Service-Oriented Architecture* o SOA), es un concepto de arquitectura de software que define la utilización de servicios para dar soporte a los requerimientos de software del usuario. Proporciona una metodología y un marco de trabajo para documentar las capacidades de negocio y puede dar soporte a las actividades de integración y consolidación.

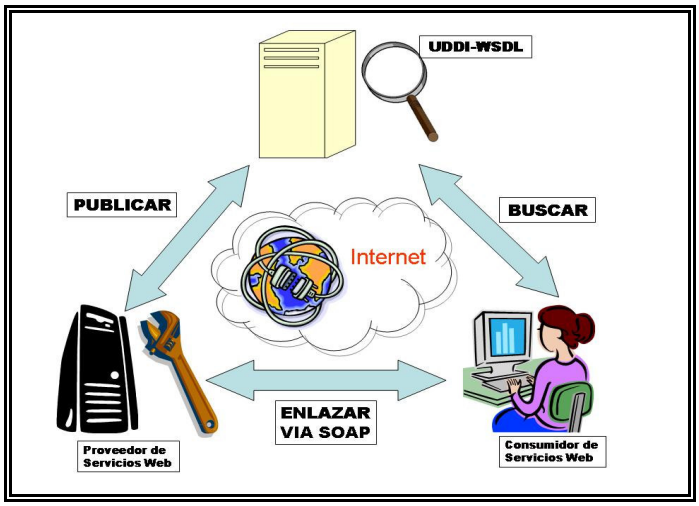

Figura 1. Arquitectura Orientada a Servicios

En la Figura 1, se observa una arquitectura orientada a servicios, en la que se distinguen los siguientes elementos principalmente: Un repositorio de servicios (UDDI), donde se encuentran descritos mediante WSDL los distintos servicios que ha publicado previamente el proveedor de Web Services. Los consumidores de los Web Services, podrán entonces consultar en el repositorio (UDDI) los servicios ofrecidos por el proveedor y si están interesados en la utilización de alguno de ellos, contactarán, vía SOAP, con las aplicaciones correspondientes que los sustentan.

En un ambiente SOA, los nodos de la red hacen disponibles sus recursos a otros participantes en la red como servicios independientes a los que tienen acceso de un modo estandarizado. La mayoría de las definiciones de SOA identifican la utilización de Web Services (empleando SOAP y WSDL) en su implementación, no obstante se puede implementar una SOA utilizando cualquier tecnología basada en servicios.

Al contrario de las arquitecturas orientadas a objetos, las SOAs están formadas por servicios de aplicación débilmente acoplados y altamente interoperables. Debido a que dichos servicios funcionan sobre diferentes tecnologías de desarrollo tales como Java y .NET, los componentes de software se vuelven muy reutilizables.

Dos ejemplos de soluciones comerciales SOA serían "Java Enterprise System" de Sun Microsystems y "Connected Services Framework" (BizTalk® Server + SQL Server + Windows Server + .NET Framework) de Microsoft.

Un ejemplo típico de arquitectura SOA son los Web Services, que proporcionan una interfaz de acceso a un servicio escondiendo las particularidades del mismo, de modo que sea accesible desde cualquier tipo de cliente a través de protocolos estándar.

b) Web Services (WS)

Internet ha introducido un entorno nuevo donde el software se va a ofrecer y acceder como servicio. Los Web Services proporcionan la plataforma ideal para poder conseguir una integración completa de los procesos de negocio de una organización con otros posibles colaboradores.

Según la W3C (el organismo que se encarga de desarrollar gran parte de los estándares de internet) se entiende por Web Service lo siguiente: "*Un Servicio Web es una aplicación software identificada mediante una URI, cuyos interfaces públicos y enlaces se definen y describen usando XML. Su definición puede ser descubierta por otros sistemas software. Estos sistemas pueden interactuar con el Servicio Web de la forma prescrita por su definición, usando mensajes basados en XML a través de protocolos estándares de internet."* [4].

En definitiva, un Web Service expone funcionalidad a un consumidor, es una URL programable y proporciona mecanismos para invocar operaciones de forma remota (a través de internet), está basado en estándares Web (HTTP, XML, SOAP, WSDL, UDDI), puede implementarse en cualquier lenguaje y en cualquier plataforma, actuando como "caja negra" (componentes reutilizable y alquilables).

Los Web Services, no son por tanto aplicaciones con una interfaz gráfica con la que las personas puedan interaccionar, sino que son software accesible en internet (o en redes privadas que usen tecnologías internet) por otras aplicaciones. De esta forma, es posible desarrollar aplicaciones que hagan uso de otras aplicaciones que estén disponibles en internet y que interaccionen con ella.

Un típico ejemplo podría ser un WS al que se le pudiese preguntar por una empresa y que nos retornase en tiempo real el valor al que están cotizando las acciones de dicha compañía. De esta forma cualquier

aplicación (ya sea Web o de escritorio) que quiera mostrar esta información sólo tendría que solicitarla a través de internet al Web Service cuando la necesitase.

Otro ejemplo de servicio Web podría ser uno que al pasarle el nombre de una ciudad, nos devolviese la temperatura, humedad, y otras condiciones climatológicas de la misma.

Básicamente los Web Services permiten que diferentes aplicaciones, realizadas con diferentes tecnologías, y ejecutándose en toda una variedad de entornos, puedan comunicarse e integrarse, lo cual es muy importante y supone un gran avance y ahorro de trabajo. Un ejemplo de esto es SpinCircuit [24], que solventa los problemas de comunicación entre el diseño de los ingenieros electrónicos y los proveedores de componentes.

Si tenemos varias aplicaciones ya desarrolladas en lenguajes propietarios o en plataformas específicas y queremos que interaccionen entre ellas, el coste de elegir a posteriori un único lenguaje o plataforma y migrarlo al mismo es descabellado en la mayoría de las situaciones, y es aquí donde los Web Services, así como otras tecnologías, pueden sernos de una grandísima utilidad.

Con los Web Services podemos reutilizar desarrollos sin importar la plataforma en la que funcionan o el lenguaje en el que están escritos. Los Web Services [25] se constituyen en una capa adicional a estas aplicaciones de tal forma que pueden interaccionar entre ellas usando para comunicarse tecnologías estándares que han sido desarrolladas en el contexto de internet.

Por lo tanto, SOA y Web Services son apropiados para aplicaciones:

- Que deben operar a través de internet, donde la fiabilidad y la velocidad no se puede garantizar.
- Donde no existe habilidad de gestionar la instalación de forma que todos los solicitantes (clientes) y proveedores se actualicen a la vez.
- Donde los componentes de un sistema distribuido se ejecuten en distintas plataformas y distintos productos.
- Donde una aplicación existente necesite exponerse para ser usada a través de la red y pueda decorarse como un Servicio Web.

2.1.1.2. Tipos de Servicios Móviles

En la actualidad existe un amplio abanico de posibilidades dentro de los servicios móviles. Dentro de ellos, encontramos unos servicios básicos, que serán tomados como reglas principales para la composición y creación de otros servicios más complejos. No sólo utilizarás tu terminal para mandar un simple mensaje de

texto sino que podrás conseguir, por ejemplo, una lista de restaurantes chinos para enviársela a tus mejores amigos.

Hemos distinguido distintos tipos de servicios, dependiendo de la complejidad de los mismos:

a) Servicios de Red

Un servicio de red es el servicio más básico que se ofrece al usuario. Algunos ejemplos son el servicio de llamada, mensajería corta (sms), llamada en espera y otros. Este tipo de servicios pueden clasificarse en las siguientes categorías:

- Servicios de tratamiento de llamadas (llamada en espera, redireccionamiento de llamada, etc.).
- Servicios de mensajería, ya sea de voz, facsímil o mensajes cortos.
- Servicios de transmisión de datos, tanto digital como analógica.
- Servicios corporativos (Red Privada Virtual RPV, conexión a centralita, etc.).
- Servicios de tarificación y costes (Prepago, facturación detallada, pago con tarjeta de crédito, consulta de facturación, límites, etc.).

b) Servicios de Información

Los servicios de información son aquellos que proporcionan al usuario los datos que ha solicitado. Entre estos servicios, encontramos, por ejemplo, un servicio de consulta meteorológica, información de tráfico, listado de establecimientos y otros. Estos servicios pueden clasificarse de diversas maneras. En función del tipo de información que ofrecen al usuario final (información deportiva, personal, ocio) o por la complejidad de los recursos necesarios para obtener y proporcionar dicha información Según la complejidad de proporcionar la información, clasificamos los servicios de información en dos tipos:

- Simples: consulta que requiere un acceso a una única base de datos. Ejemplos: información meteorológica, información de tráfico, listas de establecimientos, cartelera …
- Compuestos: consulta que requiere un acceso más complejo, necesitando acceder a varias bases de datos y manejar datos privados que requieren cierto nivel de seguridad. Ejemplos característicos, pueden ser consultar información bancaria, realizar transacciones bancarias y otros.

c) Servicios Interactivos

Un servicio interactivo es aquel que supone una interacción entre el usuario y el centro proveedor de servicios. Algunos ejemplos son: reserva de habitaciones en hoteles, compra de entradas y otros. No

destacamos una clasificación significativa dentro de este tipo de servicios, ya que existen multitud de posibilidades para clasificar todos los servicios englobados en esta categoría.

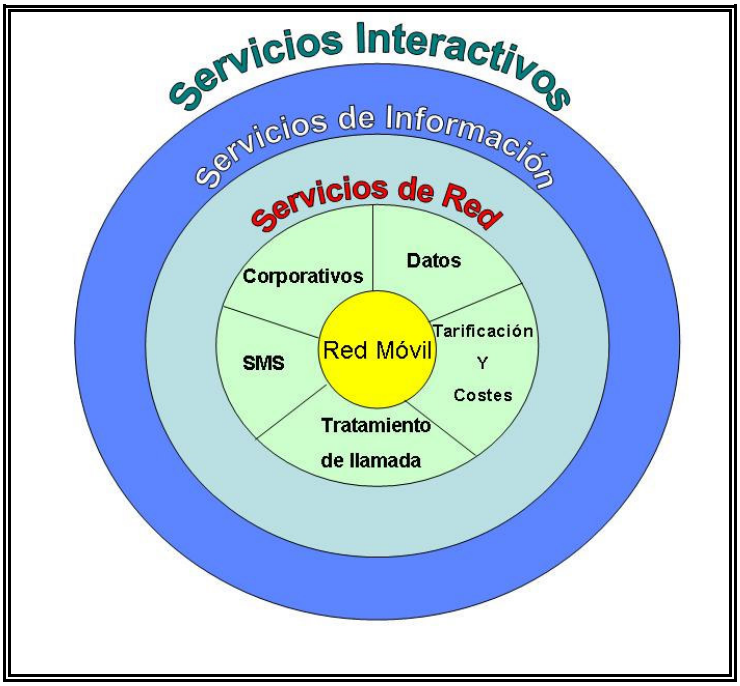

Figura 2. Clasificación de los Servicios de la Red Móvil

En la Figura 2 se aprecia la clasificación de servicios antes descrita. Se puede observar que los servicios de Red son los que están en contacto con la Red Móvil, siendo éstos los servicios más simples. Luego vienen los servicios de información que son más complejos y engloban a los servicios de red. En un nivel superior, se encuentran los servicios interactivos, que son los servicios más complejos, que engloban a todos los anteriores.

2.1.1.3. Tecnologías y Estándares Móviles

En esta sección describimos algunas de las tecnologías y estándares actuales que pueden ser utilizados para el diseño e implementación de las plataformas y los servicios anteriormente descritos.

La infraestructura mínima que requieren los Web Services se puede definir en términos de:

- Lo que va en "la red": *Formatos y protocolos de comunicación*
- Lo que describe lo que va en la red: *Lenguajes de Descripción de Servicios*

- Lo que nos permite encontrar y almacenar dichas descripciones: *Descubrimiento de Servicios*.
- Lo que permite la gestión inteligente: *control de servicios*

Las especificaciones que se han desarrollado para implementar estos mecanismos son:

- **SOAP (Simple Object Access Protocol)**: es un protocolo de aplicación basado en mensajes y que permite que una aplicación interaccione (use, instancie, ejecute, llame) al Web Service. [5]
- **WSDL (Web Service Description Languaje)**: es un formato basado en XML para definir Web Services. Describe la forma por la cual un Web Service deber ser accedido y usado. [6]
- **XML (eXtensible Markup Language)**: lenguaje que deriva de SGML, diseñado para representar y transferir datos estructurados, separa datos de formateo y transformación y se utiliza como base para definir distintos formatos.
- **UDDI (Universal Description, Discovery, and Integration):** es un repositorio de descripciones de Web Services. Es como unas "paginas amarillas" que permiten a los usuarios localizar Web Services. [7]
- **HTTP (Hypertext Transfer Protocol):** es un protocolo estándar de W3C para la transferencia de documentos en internet. Los Web Services lo utilizan como mecanismo de comunicación. Es un protocolo genérico y sin estado.
- **J2EE (Java 2 Enterprise Edition):** es un conjunto de estándares para la industria del software que interacciona con los Web Services. [20][21][22][23]
- **Tecnología de Agentes.** Los agentes inteligentes son programas informáticos en los que se delegan responsabilidades con una serie de características como son la autonomía, la reactividad, es decir, saben adaptarse y actuar en entornos que no conoce y la capacidad de percibir señales del entorno y comunicarse con otras personas o agentes.

SOAP, WSDL y UDDI forman la base de los Web Services. Las compañías crearán un nuevo Web Service y lo definirán a través de WSDL y lo almacenarán en un repositorio UDDI. Los usuarios requieren un Web Service y, dicho servicio será localizado y ofrecido al usuario. Para que sea posible la entrega del servicio, se accede a la UDDI (pública o privada) correspondiente y se ejecuta el servicio vía SOAP bajo HTTP, de acuerdo a lo especificado en la descripción WSDL.

SOAP es un protocolo estándar creado entre otros por Microsoft e IBM, que se encarga de realizar el empaquetamiento de mensajes. A diferencia de DCOM y CORBA, que son binarios, SOAP usa el código fuente en XML, que facilita la eliminación de errores, pero es menos efectivo. Estos mensajes especifican todas las reglas necesarias para ubicar servicios Web_xml, integrarlos en aplicaciones y establecer

comunicación entre servicios. Usa un lenguaje de definición de servicios llamado WSDL y corre sobre cualquier protocolo de internet, generalmente HTTP, que es el único homologado por el W3C.

¿Qué especifica SOAP?

- Describe cómo se empaqueta la información en documentos XML.
- Un conjunto de estándares para usar mensajes SOAP para definir la interacción entre clientes, es decir, definir como los clientes invocan un procedimiento remoto enviando un mensaje SOAP y como los servicios pueden responder enviando otro mensaje al llamador.
- Un conjunto de reglas para la codificación de los datos que una entidad que procesa mensajes SOAP debe seguir.
- Una descripción para el transporte de un mensaje SOAP sobre HTTP y SMTP

Formato Mensajes

Los mensajes son envoltorios que la aplicación usa para guardar la información que se quiere enviar. Esta organizado en dos partes (ver figuras 3 y 4): una cabecera (opcional) y un cuerpo (obligatorio)

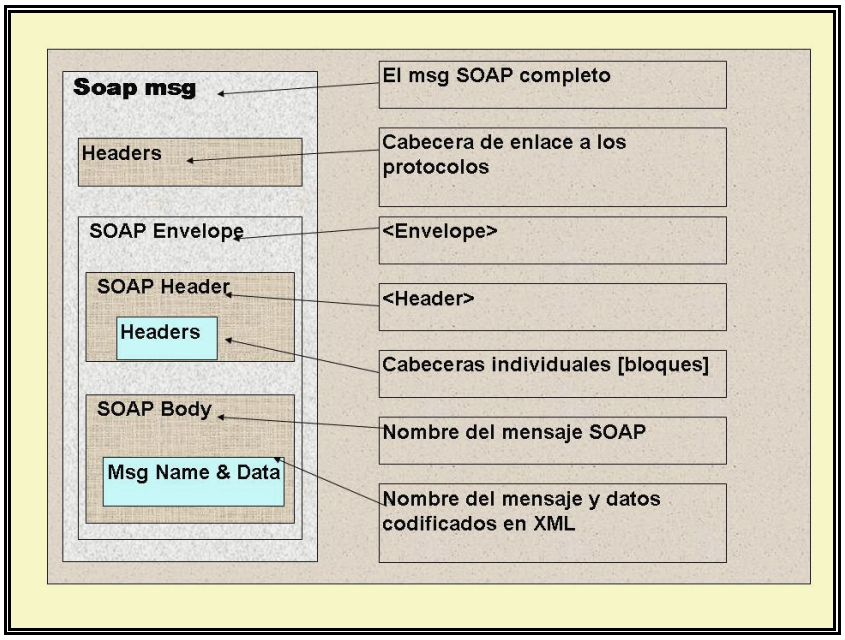

Figura 3. Formato de un Mensaje SOAP

Figura 4. Ejemplo mensaje SOAP

WSDL es un formato XML que se utiliza para describir Web Services, particularmente sus interfaces. La versión 1.1 está en estado de "propuesta de recomendación" por parte del W3C.

Describe la interfaz pública a los Web Services. Está basado en XML y describe la forma de comunicación, es decir, los requisitos del protocolo y los formatos de los mensajes necesarios para interactuar con los servicios listados en su catálogo. Las operaciones y mensajes que soporta se describen en abstracto y se ligan después al protocolo concreto de red y al formato del mensaje.

El fichero WSDL está en formato XML e indica que servicios dispones y da una breve referencia sobre ellos para luego poder invocarlos usando parámetros específicos, es decir, contiene información suficiente para hacer llamadas a Web Services.

WSDL tiene tres partes (véase Figura 5):

Las definiciones se expresan en XML e incluyen generalmente las definiciones del tipo de datos y las definiciones del mensaje que utilizan las definiciones del tipo de datos. Estas definiciones se basan generalmente sobre un cierto vocabulario XML acordado. Los vocabularios dentro de una organización se podían diseñar específicamente para esa organización. Pueden o no ser basados en un cierto vocabulario a nivel industrial. Si el tipo de datos y las definiciones del mensaje necesita ser utilizado entre las organizaciones, muy probablemente se usará un vocabulario a nivel industrial.

Las operaciones describen las acciones para los mensajes soportados por un Servicio Web. Existen cuatro tipos de operaciones:

- One-way: Mensajes enviados sin una contestación requerida.

- Petición/respuesta: El remitente envía un mensaje y receptor envía una contestación. Solicitar la respuesta: Una petición por una respuesta.
- Notificación: Mensajes enviados a los receptores múltiples.

Los "*bindings services"* conectan los puertos con su tipo de puerto. Un puerto se define mediante asociación a una dirección de red con otro tipo de puerto. Un conjunto de puertos define un servicio. Estos servicios suelen usar SOAP, pero también pueden usar CORBA, DCOM, .NET, etc.

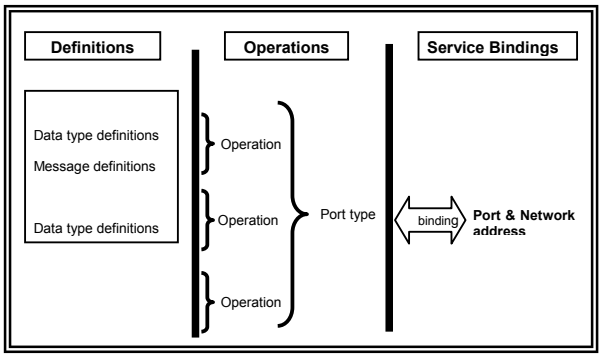

Figura 5. Estructura de un fichero WSDL

UDDI fue creado originalmente por IBM, Microsoft y Ariba, desde su versión 3 (en 2002) ha pasado a manos de OASIS (www.oasis-open.org) que a partir de ahora va a determinar su futuro y extensiones.

UDDI se concibió como un Registro de Negocio = *Servicio de Directorio y Nombrado sofisticado*. Especifica un marco para describir y descubrir Web Services.

UDDI define *estructuras de datos* y *APIs* para publicar descripciones de servicios en el registro y para consultar el registro para buscar descripciones publicadas. Las APIs de UDDI están especificadas con WSDL y con SOAP Binding, lo que permite acceder a ellas como Web Services.

La especificación UDDI tiene dos objetivos esenciales:

1. ser un soporte a los desarrolladores para encontrar información sobre Web Services y poder construir clientes.
2. facilitar el Enlace Dinámico de Web Services, permitiendo consultar referencias y acceder a servicios de interés.

Estructura de Datos de UDDI**.** La información en un registro UDDI se almacena en ficheros XML con una estructura jerárquica. Los elementos de esta estructura son:

- `businessEntity`: es el elemento "top-level", describe un negocio o una entidad que ha registrado un servicio en UDDI. Ejemplos: Departamento de Contabilidad, Servidor de Aplicaciones Corporativo. Este elemento soporta información estándar tal como nombre, descripción, e información de contacto, así como información de metadatos (por ejemplo: identificadores y categorías).
- `businessService`: describe un Servicio Web que ha sido expuesto por una entidad de negocio, soporta el nombrado de un Servicio Web y lo asocia con una entidad de negocio y con la información de *binding*. Soporta la asignación de categorías al Servicio Web (industria, productos, códigos geográficos, etc.).
- `bindingTemplate`: describe la información técnica necesaria para enlazar con un Servicio Web en particular. Este elemento soporta el nombrado de un Servicio Web y su asociación con una entidad de negocio e información de binding. La información de binding se describe como un punto de acceso que posee un atributo llamado UrlType utilizado para especificar los siete tipos de puntos de entrada: *mailto, http, Https, Ftp, Fax, Phone, Other.*
- `tModel`: (*Technology Model)*. Estructura de Metadatos Genérica para representar cualquier concepto o construcción (definiciones de protocolos, ficheros WSDL, XML schemas, Espacios de Nombres, esquemas de categorías, etc.).

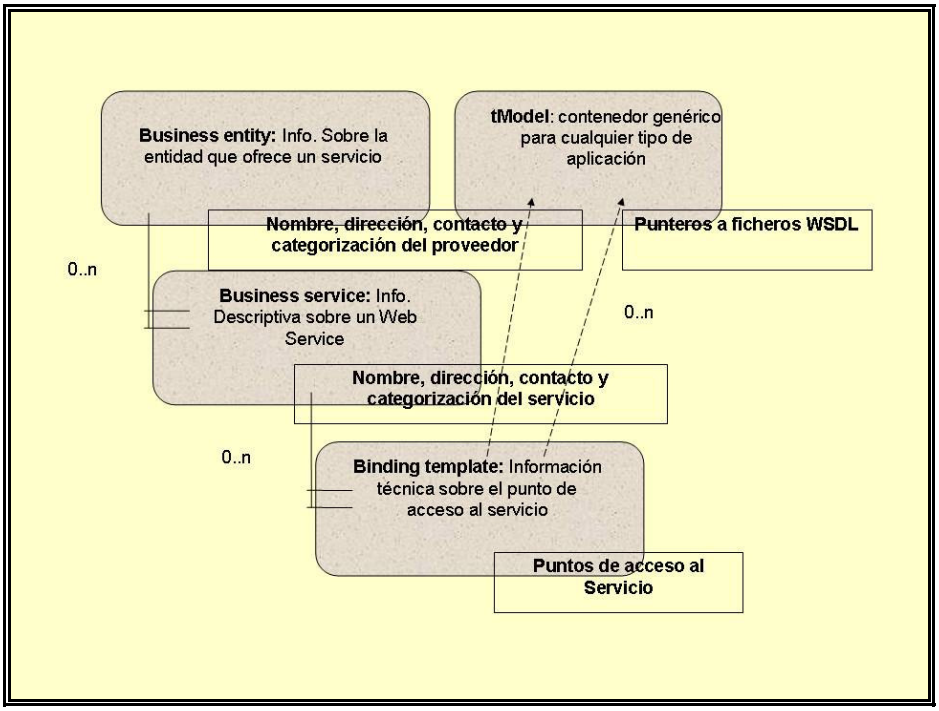

Figura 6. Estructura de UDDI

HTTP es el protocolo de la Web (WWW), usado en cada transacción. Las letras significan Hyper Text Transfer Protocol, es decir, protocolo de transferencia de hipertexto. El hipertexto es el contenido de las páginas Web, y el protocolo de transferencia es el sistema mediante el cual se envían las peticiones de acceder a una página Web, y la respuesta de esa Web, remitiendo la información que se verá en pantalla. También sirve el protocolo para enviar información adicional en ambos sentidos, como formularios con mensajes y otros similares.

HTTP es un protocolo sin estado, es decir, que no guarda ninguna información sobre conexiones anteriores. Al finalizar la transacción todos los datos se pierden. Por esto se popularizaron las cookies, que son pequeños ficheros guardados en el propio ordenador que puede leer un sitio Web al establecer conexión con él, y de esta forma reconocer a un visitante que ya estuvo en ese sitio anteriormente. Gracias a esta identificación, el sitio Web puede almacenar gran número de información sobre cada visitante, ofreciéndole así un mejor servicio.

SOAP corre sobre cualquier protocolo de internet, generalmente HTTP, que es el único homologado por el W3C. Éste protocolo es el empleado intrínsecamente para la conexión sobre internet. Garantiza que cualquier cliente con un navegador estándar pueda conectarse con un servidor remoto. La transmisión de datos se empaqueta (serializa) con XML, que se ha convertido en el "parteaguas" del intercambio de datos, salvando las incompatibilidades entre otros protocolos, tales como el NDR (Network Data Representation) o el CDR (Common Data Representation). Por otra parte, los servidores Web pueden procesar las peticiones de usuario, empleando las tecnologías de servlets, paginas ASP (Active Server Pages) o JSP (Java Server Pages), o un servidor de aplicaciones, invocando objetos de los tipos CORBA, COM o EJB.

J2EE está basado en Java y es, de hecho, un conjunto de estándares para la industria del software. Al poder interaccionar con los Web Services deberá cumplir los estándares de WSDL, UDDI y SOAP. J2EE hace posible que los Web Services sean descritos por ficheros WSDL.

Tabla 1. Tabla de Analogías y diferencias entre J2EE y .NET

Característica	.NET	J2EE
Tipo de tecnología	Producto	Estándar
Empresas que lo ofrecen	Microsoft	Más de 30
Librerías de desarrollo	. NET Framework SDK	Java core API

Intérprete	CLR	JRE
Páginas dinámicas	ASP.NET	Servlets, JSP
Componentes	.NET Managed Components	EJB
Acceso a bases de datos	ADO.NET	JDBC, SQL/J
Web Services	SOAP, WDSL, UDDI	SOAP, WDSL, UDDI
Interfaces gráficas	Win Forms y Web Forms	Java Swing
Herramienta de programación	Visual Studio .NET	Dependiente del fabricante
Transacciones distribuidas	MS-DTC	JTS
Servicios de directorios	ADSI	JNDI
Librería de encolado de mensajes	MSMQ	JMS 1.0
Lenguajes utilizados	C#, Visual Basic, C++, otros	Java
Lenguaje intermedio	IL	Bytecodes

Tecnología de Agentes: Los agentes son entidades inteligentes (como un proceso), que existen dentro de cierto contexto o ambiente y que se pueden comunicar a través de un mecanismo de comunicación inter-proceso, usualmente un sistema de red, utilizando protocolos de comunicación. Por tanto un sistema multiagente es un sistema distribuido donde la conducta combinada de dichos elementos produce un resultado en conjunto inteligente.

Los agentes Software son entidades computacionales con un comportamiento humanoide, que pueden funcionar en los equipos de los usuarios (PC, PDA, teléfonos móviles) y en los nodos de las redes, tienen autonomía y capacidad de decisión, razonan, aprenden, se comunican con otros agentes, pueden organizarse y, además pueden desplazarse de un nodo a otro.

El concepto de agente caracteriza a una entidad SW con una arquitectura robusta y adaptable que puede funcionar en distintos entornos o plataformas computacionales y, es capaz de realizar de forma “inteligente”

y autónoma distintos objetivos intercambiando información con el entorno, o con otros agentes humanos o computacionales.

Las ventajas de usar agentes son las siguientes: Mejoran la funcionalidad y la calidad, implica un menor coste, se reduce el mantenimiento facilitando la transformación y evolución y se integran adecuadamente con otras tecnologías.

La arquitectura de estos sistemas dicta como se descomponen los agentes en módulos para realizar las funciones pedidas. Se distinguen tres tipos de arquitectura:

a) Arquitectura Deliberativa. Son arquitecturas construidas a partir de una descripción simbólica del problema, se integra en el agente para, que este, razone y lleve a cabo el conjunto de tareas específicas. El problema esta en que los agentes hacen sus funciones a partir de descripciones reales. Por tanto, un agente deliberativo (o con una arquitectura deliberativa) es aquel que contiene un modelo simbólico del mundo, explícitamente representado, en donde las decisiones se toman utilizando mecanismos de razonamiento lógico basados en la correspondencia de patrones y la manipulación simbólica, con el propósito de alcanzar los objetivos del agente [Maes, 89].

b) Arquitecturas Reactivas. No se basan en ningún tipo de conocimiento simbólico sino más bien en la inteligencia, que es intrínseca al sistema. Un ejemplo de estos sistemas son los agentes aplicados en robótica.

c) Arquitecturas Híbridas. Combinación de sistemas deliberativos (a partir de símbolos) y reactivos (generan acciones).

Un sistema inteligente adaptativo, dentro de las arquitecturas híbridas, es un sistema experto que razona e interactúa con su entorno en tiempo real. Tiene dos niveles, uno físico (que efectúa los trabajos relacionados con la coordinación entre percepción y acción, tales como interpretar, filtrar y reaccionar al ambiente dinámico en el cual el agente esta embebido) y otro cognitivo (que recibe información del nivel físico, construye un modelo evolutivo del mismo e interpreta, razona y planifica a partir de él).

2.1.2. Introducción a las Técnicas de Inteligencia Artificial

En la actualidad existen muchas posibilidades para definir la Inteligencia Artificial dependiendo principalmente del campo donde se aplique esta técnica. Entre las principales definiciones podemos destacar las siguientes: *"El arte de crear máquinas con capacidad de realizar funciones que realizadas por personas requieren de inteligencia."* [Kurzweil, 90]. *"El estudio de cómo lograr que las computadoras realicen tareas que, por el momento, los humanos hacen mejor."* [Knight, 91]. *"La rama de la ciencia de la computación que se ocupa de la automatización de la conducta inteligente."* [Luger y Stubblefield, 93].

Observando estas definiciones se puede llegar a entender lo importante que será dotar a un sistema orientado a servicios móviles de cierto tipo de inteligencia.

En el campo de la Inteligencia Artificial podemos diferenciar mayoritariamente dos técnicas: la que está basada en el conocimiento y la que está basada en el comportamiento.

El razonamiento basado en conocimiento resuelve los problemas en el dominio del cual posee un conocimiento específico. Además se encarga de buscar la solución al problema que más se asemeje a la respuesta que daría un humano independientemente del proceso seguido.

En cambio, el razonamiento basado en comportamiento es aquel que reacciona a partir de acciones construyendo su conocimiento para adaptarse a los distintos estímulos o cambios que se produzcan.

2.1.2.1. Sistemas Inteligentes Basados en Conocimiento

Los Sistemas Inteligentes Basados en Conocimiento (KBAI) poseen una arquitectura que refleja de alguna manera la estructura del conocimiento y de los procesos humanos. Por un lado está la memoria (largo plazo) en la que se guardan los hechos del sistema (base de hechos) y los conocimientos (base de conocimiento) acerca del dominio del que tienen experiencia. Por otro lado esta la parte del sistema que realiza las funciones del razonamiento para la resolución de problemas (motor de inferencia). Por último tendremos unidades para la comunicación del sistema con su entorno (unidades de entrada/salida). (Véase Figura 7).

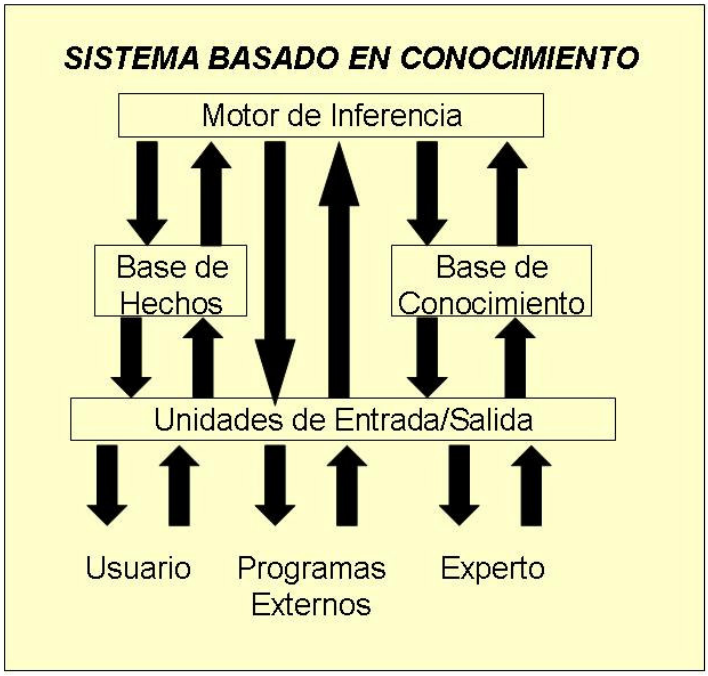

Figura 7. Sistema Basado en Conocimiento

Dentro de este tipo de sistemas podemos distinguir los siguientes:

a) Sistemas Expertos (sistemas de Razonamiento Basado en Reglas)

Son aquellos que trabajan sobre un dominio concreto y basan su éxito o su fracaso en dicho conocimiento y en su capacidad de aprendizaje, es decir, que emulan el comportamiento de un experto en un campo de aplicación delimitado. El conocimiento sobre el dominio proporciona al sistema experto mayor información sobre el problema a tratar y su entorno, de forma que pueda generar y adaptar soluciones de forma más precisa, al tener un conocimiento más profundo sobre el tema, de forma similar a un experto especializado. El aprendizaje, inductivo o deductivo según corresponda, proporcionará al sistema experto de mayor autonomía a la hora de abordar problemas totalmente desconocidos, pudiendo generar nuevo conocimiento partiendo del extraído inicialmente del experto o expertos humanos.

Los sistemas expertos están más generalizados y pueden implementarse en múltiples lenguajes de programación, tales como java, prolog, etc. Estos sistemas expresan el conocimiento inferido mediante árboles de decisión. Como en cualquier otro sistema, cuanto más amplio sea el conocimiento de partida, mejor será el resultado del experto.

b) Redes Semánticas

Son sistemas de representación del conocimiento basados en una red de nodos y enlaces. Los nodos representan conceptos o significados mientras que los enlaces relacionan dichos conceptos. Una de las características más importantes a destacar de estos sistemas es la herencia por defecto. Esta propiedad se define como "*el sistema de razonamiento que lleva a un agente a deducir propiedades de un concepto basándose en las propiedades de conceptos más altos en la jerarquía.*" [Shastri, 88]

Básicamente podemos distinguir tres categorías de redes semánticas:

- Redes ES-UN, en las que los enlaces entre nodos están etiquetados con la relación "es_un".
- Grafos conceptuales, en los que existen dos tipos de nodos: conceptos y relaciones.
- Redes de Macros, en los que los puntos de unión de los enlaces son parte de la etiqueta del nodo.

c) Lenguaje Natural

"Es un lenguaje escrito o hablado usado por una comunidad que es precisamente lo contrario a un lenguaje para establecer comunicación con una computadora, mediante la entrada de datos o la programación" [Guzmán, 97]

La comprensión y reconocimiento del lenguaje natural es uno de los problemas más complejos a los que se enfrenta la inteligencia artificial debido a la complejidad, regularidad y diversidad del lenguaje humano y a los problemas de interpretación asociados a las frases, oraciones y textos.

El procesamiento del lenguaje es de manera general, el conjunto de instrucciones que una computadora recibe en un lenguaje de programación dado, que le permitirán comunicarse con un humano en su propio lenguaje. Por tanto el procesamiento del lenguaje natural tiene como objetivo fundamental lograr una comunicación máquina-humano similar a la comunicación humano-humano.

Puede ser establecido o desarrollado dentro de una ontología (conjunto de entidades relevantes en el campo de una aplicación determinada, así como las interacciones entre las mismas).

2.1.2.2. Sistemas Inteligentes Basados en Comportamiento (BBAI)

Los Sistemas Inteligentes Basados en Comportamiento (BBAI) fueron desarrollados originalmente por Brooks en 1986. En BBAI, la inteligencia se compone de una gran cantidad de elementos modulares que son relativamente simples de diseñar. Cada elemento opera solamente en un contexto particular, que el módulo en sí mismo reconoce. En la propuesta original de Brooks, estos módulos son máquinas finitas de estados, organizadas en capas. Los comportamientos no tienen ningún acceso a otro estado interno, pero pueden supervisar y/o alterar cada entrada y salida. Las capas son una abstracción de organización: los comportamientos de cada capa alcanzan una de las metas del agente, pero una capa más alta puede incluir la

meta de una capa más baja a través del mecanismo que afecta a las entradas y a las salidas. Los sistemas basados en comportamiento han logrado un gran alcance debido a la robustez y la simplicidad de los programas.

Dentro de sistemas BBAI podemos distinguir los siguientes:

a) Aprendizaje Automático

Esta técnica permite automatizar la planificación y toma de decisiones de procesos y tareas críticas para el sistema. Introduce gran dinamismo en los sistemas informáticos al dotarles de capacidad de adaptación "automática" a los cambios de necesidades y perfiles de usuarios.

Estos sistemas son limitados ya que no podrán resolver problemas para los que no hayan sido programados. Es decir, sus límites estarán impuestos según el conocimiento integrado en ellas. Por eso de hecho, sería más útil e inteligente una aplicación capaz de adaptarse y poder integrar nuevo conocimiento, de manera que así pueda resolver nuevos problemas.

El área de aprendizaje automático es muy amplia y ha dado lugar a muchas técnicas diferentes de aprendizaje, tales como Aprendizaje inductivo (generalización), deductivo, etc.

b) Agentes. Sistemas Basados en Acciones

Los agentes son entidades *inteligentes* (como un proceso), que existen dentro de cierto contexto o ambiente y que se pueden comunicar a través de un mecanismo de comunicación inter-proceso, usualmente un sistema de red, utilizando protocolos de comunicación. Por tanto, un sistema multiagente es un sistema distribuido donde la conducta combinada de dichos elementos produce un resultado en conjunto *inteligente*.

Un agente, tal como ha quedado explicado anteriormente, puede ser usado en diferentes aplicaciones, tales como la ayuda al cliente, búsqueda de información personalizada, etc.

Dentro de la definición de agente entra el concepto de robot autónomo [Brooks, 86] y de los agentes que se mueven en mundos artificiales, los cuales sienten su entorno, actúan sobre él, cambian de estado interno, aprenden y evolucionan; sin embargo estos agentes únicamente pueden ser capaces de simular aspectos del mundo real para así tratar de entenderlo mejor.

c) Mecanismo de Selección de Acciones

El Mecanismo de Selección de Acciones (ASM) [Maes, 89] es aquel que elige la mejor acción de un conjunto de acciones dadas en una situación y que además conoce la actividad o comportamiento desencadenante de esa acción. La acción seleccionada será aquella que llegue más satisfactoriamente al

objetivo, mientras que al mismo tiempo esté atento a las demás acciones que haya, pero descartando las soluciones para otros objetivos activos.

El camino que un agente lleve para ejecutar la acción seleccionada depende en gran medida de los objetivos que tengamos concretados. Un agente, normalmente, tiene objetivos contradictorios de varios tipos [Maes, 96]. Éste mismo, definirá "objetivos finales" (a los que quiere llegar) y "objetivos negativos" (los que debe evitar). Existen dos tipos de objetivos, los implícitos y los explícitos. Los implícitos son aquellos que no pueden cambiar a menos que el agente los reprograme; mientras que los explícitos son aquellos que existen en agentes complejos y varían en el tiempo.

Toby Tyrrell realizó una serie de pruebas al mecanismo de selección de acciones de Maes en un entorno simulado, basándose en el comportamiento de los animales y probó que tenía deficiencias en el diseño del modelo ya que muchos de los aspectos del diseño de este mecanismo no son capaces de comportarse como una persona a la hora de resolver problemas seleccionando acciones.

d) Subsumption [Brooks, 86]

Rodney Brooks fue el pionero que presentó esta alternativa, la cual consiste en llevar a cabo las tareas de conducta en capas [Jen, 98]. El comportamiento es implementado como una maquina finita de estados organizada jerárquicamente en capas, la cual producirá acciones tanto de salida como de entrada. En la arquitectura de "Subsumtion" creada por Brooks el comportamiento de las capas compiten por el control total. Los niveles más bajos son los comportamientos más concretos, comúnmente tiene preferencia frente a los niveles más altos, que incluyen los comportamientos más abstractos [Woo, 05] [Jen, 98]. Brooks implementó un robot controlado por el modelo "subsumption", donde las capas más bajas incluyen obstáculos a evitar y las capas más altas incluyen rutas más complicadas de planificar. En concreto, es evidente que los niveles más bajos deberán incluir mecanismos para la detección de obstáculos para que, por ejemplo, el robot no choque contra la pared.

La arquitectura de Brooks "Subsumtion" tiene las siguientes propiedades a destacar:

- Con respecto a la propia Arquitectura:
  - Máquina finita de estados mínima ("hormones" mecanismo para hacer mínimo el conjunto de estados) por cada capa
  - Acoplamiento entre sensores, activadores y hardware.
- Con respecto a los Agentes:
  - Sensores basados en la conducta.

- Descomposición jerárquica del comportamiento.
- Comportamiento robusto. A diferencia de otras arquitecturas, ésta demuestra solidez incluyendo determinados métodos efectivos para evitar obstáculos y otras medidas para su supervivencia.
- Representación global del estado mínima.
- No incluye símbolos
- Reacción en tiempo real
- Preconexión a patrones de conducta.
- Tasa de procesamiento lento
- Control distribuido

En resumen, la arquitectura "subsumption" esta diseñada para una limitada parte del mundo real, en particular, robots de navegación y detección de obstáculos.

2.1.2.3. Sistemas Híbridos

Los sistemas híbridos combinan aspectos deliberativos y reactivos. Los reactivos se encargan de procesar los estímulos que no necesitan deliberación mientras que los deliberativos determinan qué acciones deben realizarse para satisfacer los objetivos locales y la cooperación con los demás agentes (véase punto 2.1.1.3, tecnología de agentes).

Una arquitectura híbrida está compuesta por un conjunto de tres capas [Jen, 98]: una capa reactiva, capa de conocimiento y por último una capa de conocimiento "social". La capa más baja suele ser la capa reactiva, donde dicta en qué se basan los sensores de datos del sistema, esta capa es implementada de forma muy similar a la arquitectura "subsumption" [Brooks, 89]. La capa del medio (conocimiento) realiza una abstracción de los datos suministrados por la capa reactiva usándose para ello una representación simbólica (deliberativa). Y por último, la capa más alta es la que mantiene el contacto con los demás agentes del sistema (de ahí el nombre de social).

Ejemplos de este tipo de sistemas son aquellos que combinan:

- **Tecnologías de aprendizaje**: como por ejemplo el razonamiento basado en casos, que es una parte de la IA que se utiliza para la resolución de problemas. Se basa en problemas similares ocurridos en el pasado, denominados casos, para encontrar soluciones a los mismos, modificar soluciones existentes y explicar situaciones análogas. Frente a otros campos de IA, estos sistemas de

razonamientos son capaces de valerse de experiencias previas para llevar a cabo la resolución de un problema, para ello capturan las características de dicho problema, buscan casos similares, analizan las posibles soluciones, proponen soluciones nuevas y, por último, aprenden del problema actual para futuros problemas.

- **Lógica Difusa** (también llamada Fuzzy logic): rama de la IA que maneja los conceptos "vagos", es decir aquellos que nuestro cerebro esta acostumbrado a tratar. Por tanto emula el razonamiento humano.
- **Redes Neuronales Artificiales:** intentan emular el comportamiento humano almacenando la información en patrones, usándolos y resolviendo problemas con ellos. Para ello una red neuronal artificial puede verse como un conjunto de neuronas interconectadas de tal forma que la salida de una cualquiera sirve generalmente como la entrada de otras. La principal propiedad de este tipo de redes es la capacidad de aprender (adaptarse) del entorno en el que opera y mejorar su funcionamiento.

Otro tipo de técnicas a destacar en este sector híbrido, son aquellas que combinan un algoritmo genético con agentes software autónomos (véase punto 2.1.1.3, Tecnología de Agentes). Es decir, un sistema inteligente adaptativo que consta de una parte física, que se comunica con los estímulos del exterior y otra parte con conocimiento, que basa sus acciones en lo recogido por la capa física y a partir de ellas, interpreta y razona. Concretamente un sistema experto que razona y se comunica con su entorno en tiempo real.

2.1.3. Selección de Arquitecturas, Servicios y Tecnologías

2.1.3.1. Visión Global de la Arquitectura

La Arquitectura mostrada en la siguiente figura, está basada en la idea original de la arquitectura propuesta por nuestra profesora tutora del proyecto (Arquitectura Mes, Motor de registro y entrega de servicios), en la que se observa la presencia de SIOS-m, dividido principalmente en dos módulos, que es el motor de registro y entrega de servicios de la arquitectura.

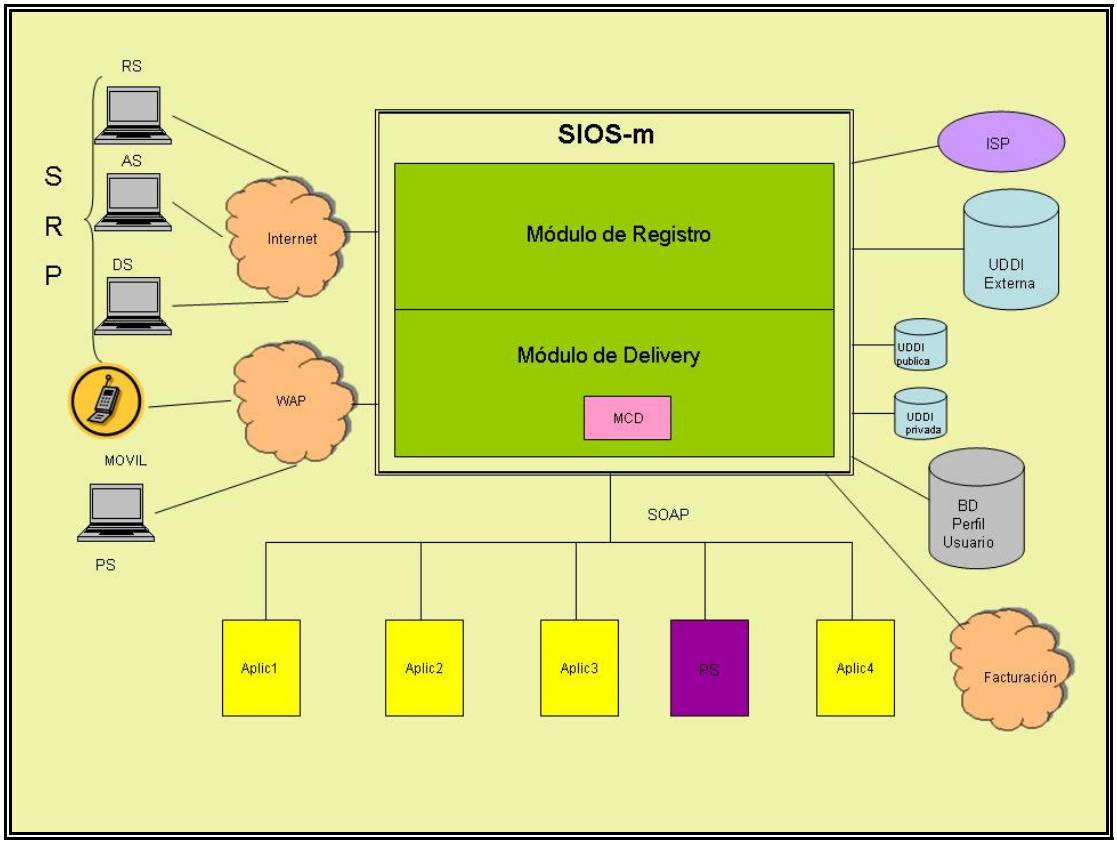

Figura 8. Visión Global de la Arquitectura.

El esquema mostrado en la Figura 8 muestra la interconexión entre los diferentes elementos de la arquitectura.

1. Usuarios finales: Dispositivos móviles, proveedores de servicios (ISP, NSP)
2. Repositorios de servicios (UDDI): donde se encuentran descritos los servicios ofertados por el proveedor. Podemos distinguir la existencia de una UDDI interna privada y una UDDI interna pública.

   La UDDI interna privada es la que contiene la descripción de los servicios accesibles únicamente por el proveedor de servicios, mientras que la interna pública contiene la descripción de los servicios accesibles desde otros proveedores de servicios.

   Existe una UDDI externa que engloba las UDDI internas públicas de los proveedores de servicios.

3. Los propios servicios descritos en ficheros XML (WSDL) almacenados en la UDDI del proveedor, que se ejecutaran a través de las aplicaciones correspondientes.

   Los servicios descritos en el repositorio, pueden clasificarse según su complejidad en:

   - Servicios Básicos: son los ofrecidos por el propio proveedor. Serán tomados como "reglas" para la composición y creación de otros servicios más complejos.
   - Servicios Compuestos: son ofrecidos también por el propio proveedor, pero están formados por la combinación de distintos servicios básicos.
   - Servicios Exclusivos de otro PS: servicio ofrecido por otro proveedor pero registrado en el repositorio de nuestro proveedor.

   La clasificación vista en el punto 2.1.4.1 engloba todos los servicios disponibles en el proveedor de servicios, siendo posible que un servicio de red sea un servicio básico o compuesto, igual que un servicio de información o un servicio interactivo.

4. Las propias aplicaciones interconectadas con el proveedor a través del protocolo SOAP que implementan los servicios descritos.
5. Una base de datos que almacena información sobre los usuarios que han usado los diferentes servicios.
6. SIOS-m, dentro del cual distinguimos dos módulos principales:

   - Módulo de Registro, encargado de gestionar los servicios
   - Módulo de Entrega de Servicios (Delivery), encargado de entregar al usuario final los servicios que haya solicitado. Dentro de éste, encontramos un módulo de control (MCD), que será el cerebro inteligente de SIOS-m.

7. El proveedor de servicios requiere una aplicación que lleve a cabo la facturación de los servicios.
8. Podemos distinguir a los SRP (*Services Repository People*), que son los encargados de registrar y administrar los servicios del proveedor. En los SRP y atendiendo a la clasificación explicada posteriormente, podemos distinguir distintos roles, cuya misión, aparte del correcto funcionamiento del módulo de registro, será dotar al módulo de registro del conocimiento que necesita.

   Dentro de los SRP podemos diferenciar tres departamentos que se encargarán de los tres diferentes tipos de servicios que previamente se han definido:

   - Un primer departamento encargado de todo lo referente a los servicios que denominamos básicos, en el que encontramos los siguientes roles:

- ❖ Arquitectos de servicios básicos: son los encargados de "inventar" y diseñar nuevos servicios básicos. Sobre este rol recaerá el éxito o fracaso de aceptación de este servicio por parte de los usuarios del Proveedor.
- ❖ Descriptores de servicios básicos: dotan al módulo de registro de la información más descriptiva del nuevo servicio. Se entiende como información descriptiva aquella que abarca aspectos acerca de la descripción técnica en formato WSDL del servicio. Son programadores encargados de traducir el servicio a un seudo código que el módulo de registro entiende y convierte a formato WSDL para el posterior volcado al repositorio de servicios. Existe la posibilidad que este rol sea el encargado de traducir directamente a formato WSDL el propio servicio, en el caso de que el módulo de registro no realice la traducción.
- ❖ Administrador de servicios básicos: encargados de gestionar los servicios básicos disponibles en el proveedor, viendo qué servicios son los más utilizados por los usuarios, cuáles los menos, qué servicios hay que promocionar más, cuáles hay que activar o desactivar, qué ofertas hay que ofrecer al usuario para los distintos servicios, etc.
- ❖ Supervisores básicos del Repositorio UDDI: comprueban que la descripción del servicio básico ha sido introducida correctamente en su lugar correspondiente en el repositorio. Supervisan el correcto funcionamiento de todo lo relacionado con la parte del repositorio donde se encuentra este tipo de servicio.

- Un segundo departamento encargado de todo lo referente a los servicios que denominamos compuestos, en el que encontramos los mismos roles que el departamento anterior con ligeras modificaciones acorde con el nuevo tipo de servicio:
  - ❖ Arquitectos de servicios compuestos: encargados de diseñar la posible combinación de servicios básicos para formar nuevos servicios que sean atractivos para los posibles usuarios.
  - ❖ Descriptores de servicios compuestos: tienen la misma tarea que los descriptores de servicios básicos. Deben comprobar posibles optimizaciones a la hora de juntar los dos seudo códigos de los servicios básicos o de acoplar de manera adecuada las dos descripciones WSDL de los correspondientes servicios según proceda.
  - ❖ Administrador de servicios compuestos: encargados de gestionar los servicios compuestos disponibles en el proveedor, viendo cuáles son los más utilizados por los usuarios, cuáles los menos, qué servicios hay que promocionar más, cuáles hay que activar o desactivar, qué ofertas hay que ofrecer al usuario para los distintos servicios, etc.

- ❖ Supervisores compuestos del Repositorio UDDI: comprueban que la descripción del servicio compuesto ha sido introducida correctamente en su lugar correspondiente en el repositorio. Supervisan el correcto funcionamiento de todo lo relacionado con la parte del repositorio donde se encuentra este tipo de servicio.

- Un tercer departamento encargado de todo lo referente a los servicios que denominamos exclusivos de otro proveedor de servicios, en el que encontramos:

  - ❖ Arquitectos de servicios exclusivos: para este tipo de servicios los arquitectos tienen que estudiar qué servicios de otros proveedores de servicios (externos) son interesantes para utilizar en nuestro propio proveedor. Tienen que supervisar la correcta comunicación vía SOAP con el otro proveedor. Son los encargados de que este servicio exclusivo se comporte de manera análoga a los propios del Proveedor.

  - ❖ Descriptores de servicios exclusivos: supervisan el seudo código entregado por el otro proveedor o la descripción WSDL del servicio según proceda.

  - ❖ Administrador de servicios exclusivos: encargados de gestionar los servicios exclusivos disponibles en el proveedor, viendo cuáles son los más utilizados por los usuarios, cuáles los menos, qué servicios hay que promocionar más, cuáles hay que activar o desactivar, qué ofertas hay que ofrecer al usuario para los distintos servicios, etc.

  - ❖ Supervisores exclusivos del Repositorio UDDI: comprueban que la descripción del servicio compuesto ha sido introducida correctamente en su lugar correspondiente en el repositorio. Supervisan el correcto funcionamiento de todo lo relacionado con la parte del repositorio donde se encuentra este tipo de servicio.

Hay que destacar que el papel del rol de Descriptor de Servicios tendrá asociadas distintas tareas según, el módulo de registro sea capaz o no, de traducir él mismo el servicio a un formato WSDL o si esta tarea es propia de este rol.

Por encima de los tres departamentos, podemos distinguir a un Arquitecto General de Servicios, que controla a todos los Arquitectos, un Descriptor General de Servicios, que controla a todos los Descriptores, un Administrador General de Servicios, que controla a todos los Administradores y un Supervisor General de Servicios, que controla a todos los Supervisores. Y por último, encontramos al Mánager del proveedor de servicios, que es el responsable del correcto funcionamiento de los servicios ofrecidos por el

proveedor.

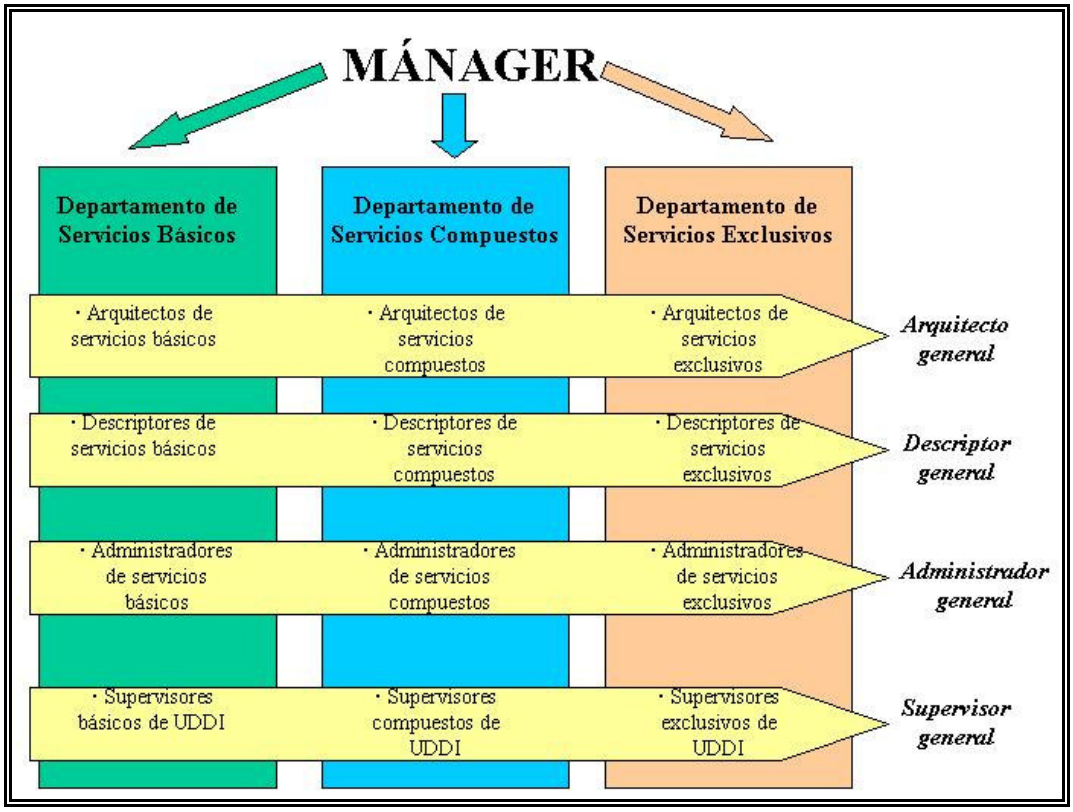

Figura 9. Estructura jerárquica de los SRP

2.1.3.2. Módulos de Registro de Servicios

Este es el módulo de interconexión entre los *Services Repository People* y el Repositorio.

- Si "*Services Repository People*" describe el servicio ya en formato WSDL, el módulo de registro será el encargado de tratarlo para almacenarlo correctamente en la UDDI.
- Si "*Services Repository People*" describe el servicio en un formato diferente, el módulo de registro será el encargado de pasarlo a WSDL y almacenarlo en la UDDI.

Entre sus tareas principales están:

- Crear uno nuevo.
- Modificar un servicio.
- Borrar alguno ya existente.

- Activar/Desactivar un servicio que no queremos que sea visible por el usuario. Es decir, almacenado en el repositorio, pero no accesible.

Para realizar este tipo de actividades de registro, sería especialmente interesante que la plataforma SIOS-m dispusiera de un modelo de "inteligencia" basado en conocimiento. Por ejemplo, un sistema que requiere almacenar información sobre muchos y muy diversos tipos de servicios necesita estar dotado de inteligencia, desde el punto de vista de almacenamiento, gestión y control de información. Y para este tipo de sistemas, existen a día de hoy diversas posibilidades, muy utilizadas en lo que se conoce como IA clásica (convencional) orientada al conocimiento (KBAI). En la sección 2.1.6 mostramos el resultado de un estudio de viabilidad de técnicas KBAI para la posible implementación de éste módulo.

2.1.3.3. Módulos de Entrega de Servicios

Este es el módulo que dota de un nuevo tipo de inteligencia a SIOS-M. Dicha inteligencia estará basada en el comportamiento, es decir será adaptativa. Encargado del control y mantenimiento de las aplicaciones que hacen que los servicios funcionen.

Dentro de este módulo existirá un "cerebro (agente) inteligente autónomo" que, aprovechando las opciones de la tecnología de agentes aplicables, extenderán o mejorarán la interacción con los usuarios.

Entre sus tareas principales están:

- Solicita servicio: recoge de la UDDI el XML donde esta cómo está descrito el servicio y cómo debe ser accedido.
- Accede Aplicación (Envío): se comunica vía SOAP con la aplicación correspondiente al servicio solicitado de acuerdo a lo "dictado" en el XML.
- Responde Aplicación: vía SOAP devuelve al módulo de entrega, los datos resultado del servicio solicitado.
- Entrega servicio: devuelve al usuario final el resultado del servicio pedido.
- Componer: haciendo uso de los resultados de otros servicios, y conforme al contenido del XML, los combina para dar como resultado al servicio solicitado.
- Registro Usuario: almacenar en la BD "Perfiles de Usuario" la información relativa al usuario que solicita usar el servicio, para su posterior uso.
- Contabilizar: llevar ciertas estadísticas del uso de los servicios por parte de los usuarios (servicios que más se usan, ...)
- Facturar: llevar la cuenta del uso de diferentes servicios.

Para realizar este tipo de actividades de entrega, sería especialmente interesante que la plataforma SIOS-m dispusiera de un módulo de "inteligencia" basado en comportamiento debido a su contacto directo con el usuario final. Por tanto tendrá que ser capaz de construir y actualizar distintos perfiles de usuario y utilizarlos para mejorar la funcionalidad que presta a los mismos. Necesita aprender de la interacción del usuario con el sistema, necesita conocer los hábitos del usuario, gracias a su perfil, consiguiendo anticiparse a determinadas peticiones del mismo, sugerirle tareas rutinarias o incluso corregir posibles errores. Gracias a esto, se consigue un proceso de interacción sencillo y robusto.

2.1.4. Modelación de un Entorno Genérico de Operación

Basándonos en la Visión Global de la Arquitectura, realizada en el punto 2.1.3.1 (véase Figura 8), para modelar nuestro entorno de operación, elegimos los elementos mostrados a continuación.

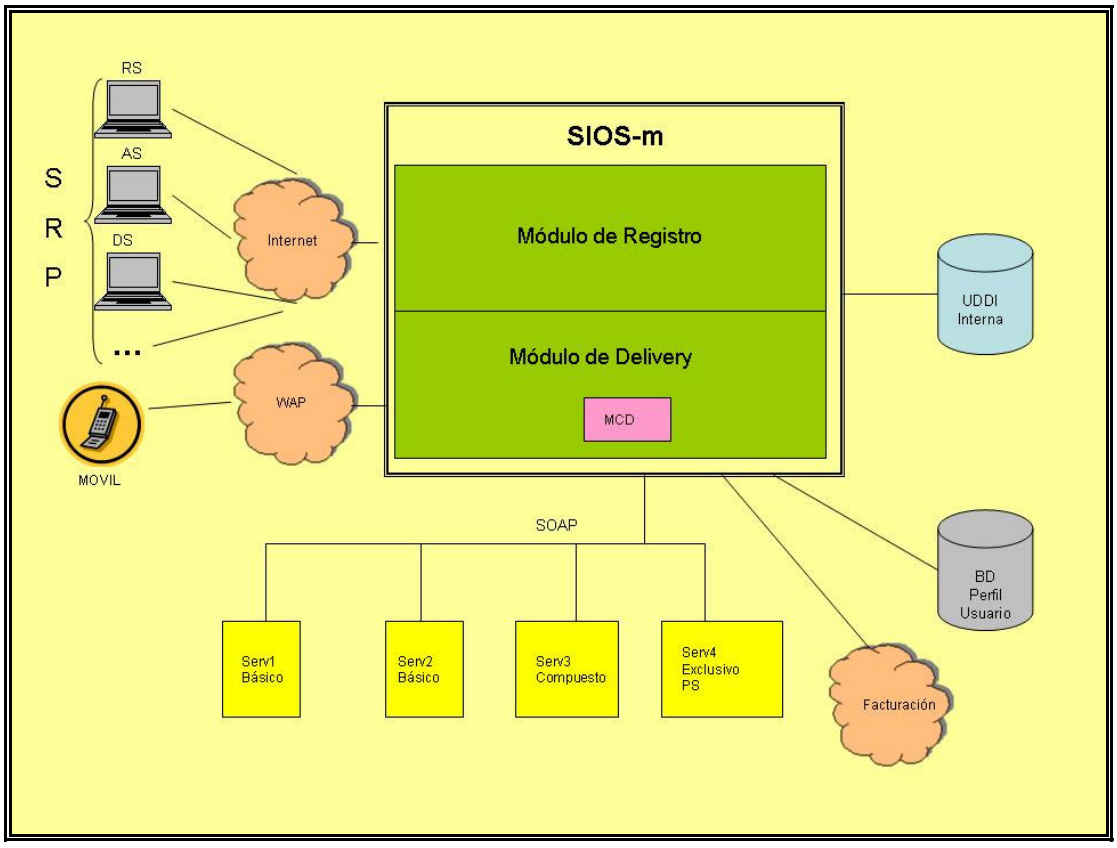

Figura 10. Entorno Genérico de Operación.

La Figura 10 muestra los siguientes elementos:

1. Usuarios Finales: de todos los posibles usuarios hemos elegido los dispositivos móviles.
2. Un repositorio de servicios (UDDI) donde se encuentran descritos los servicios ofertados por nuestro proveedor.

   Dentro del repositorio encontramos descritos cuatro servicios. Dos servicios básicos, uno compuesto y un servicio exclusivo ofrecido por otro proveedor.
3. Cuatro aplicaciones que sustenten los servicios descritos en el repositorio.
4. Contemplamos la posibilidad de tener una base de datos donde se almacenan los perfiles de usuario que usen nuestros servicios.
5. SIOS-m, donde distinguimos dos módulos principales:
   - Módulo de Registro, encargado de gestionar los servicios.
   - Módulo de Entrega de Servicios (Delivery), encargado de entregar al usuario final los servicios que haya solicitado). Dentro de éste, encontramos un módulo de control (MCD), que será el cerebro inteligente de SIOS-m.
6. El proveedor de servicios requiere una aplicación que lleve a cabo la facturación de los servicios.
7. Los SRP son los mismos que se contemplan en la visión global de la arquitectura (Véase punto 2.1.3.1)

2.1.4.1. Tipos de Usuarios

Podemos distinguir dos tipos de usuarios dependiendo del módulo con el que interactúen.

Si interactúan con el módulo de registro estaremos hablando de lo que anteriormente definimos como *Services Repository People*.

En cambio, si el módulo con el que se comunica es el de Entrega de Servicios (Delivery) y viendo el abanico de posibilidades de los distintos tipos de usuarios, hemos optado por trabajar únicamente con los dispositivos móviles.

La diferencia fundamental entre estos dos tipos de usuarios radica en el que unos (Services Repository People) son los responsables de registrar y sustentar los distintos servicios ofrecidos por la plataforma, mientras que los otros (dispositivos móviles) son los usuarios finales que disfrutan de dichos servicios.

2.1.4.2. Dispositivos Móviles

En esta arquitectura, los usuarios finales pueden acceder a la plataforma utilizando cualquier tipo de dispositivo móvil. En la actualidad, la mayoría de las plataformas son configuradas desde sus portales de acceso para poder dar servicio a PDAs, móviles de última generación, etc. SIOS-m puede integrarse con cualquier portal Web por lo que, en principio, podría ser utilizado con cualquier tipo de dispositivo móvil.

La evolución sufrida en el campo de la telefonía móvil ha permitido que en la actualidad existan multitud de teléfonos con características increíbles, con pantallas de grandes dimensiones capaces de representar imágenes de gran resolución. Cada tipo de teléfono móvil tiene un tamaño de pantalla distinto y una resolución de pantalla diferente, por lo que deducimos que será necesario incluir un módulo de control de pantallas dentro de nuestro módulo de Entrega, para así mostrar el mismo contenido en diferentes configuraciones de pantallas de teléfonos móviles.

2.1.4.3. Tipos de Servicios

Dentro del Proveedor de Servicios donde se encuentra SIOS-m tenemos dos servicios de red básicos simples, que pueden ser combinados para formar un tercer servicio compuesto. Además, estaríamos interesados en usar un servicio exclusivo ofrecido por otro PS.

Tenemos una UDDI interna privada donde se describen nuestros servicios y accederemos a la UDDI interna pública del otro PS para conseguir el servicio exclusivo.

2.1.4.4. Repositorio de Servicios

El Repositorio de Servicios es el módulo donde se almacenan los servicios. Este elemento de la plataforma actúa como una base de datos y puede ser implementado con tecnologías Web Services (en concreto, UDDI).

SIOS-m ha sido diseñado para incorporar uno y varios registros UDDI (tantos como sean necesarios). En la implementación final que mostramos en este documento (véase Fase II), los SRP describen y manipulan los servicios del entorno con ficheros WSDL.

Es accedido tanto por los *Services Repository People* del modulo de registro, para administrar los servicios, como por el módulo de Delivery para el control de la ejecución de los servicios.

2.1.4.5. Módulos de Registro y Entrega de Servicios

Ambos módulos están basados en una arquitectura software J2EE, ver Figura 11, compuesta por tres capas: una capa de presentación (Interfaz grafica), a través de la cual se comunica con los usuarios, una capa de Lógica de Negocio en la que se encuentra la funcionalidad de cada módulo y una capa de acceso a los datos que se encuentran tanto en una base de datos, tipo JDataStore donde tenemos los perfiles de usuario, como en un repositorio UDDI donde tenemos las descripciones de los servicios.

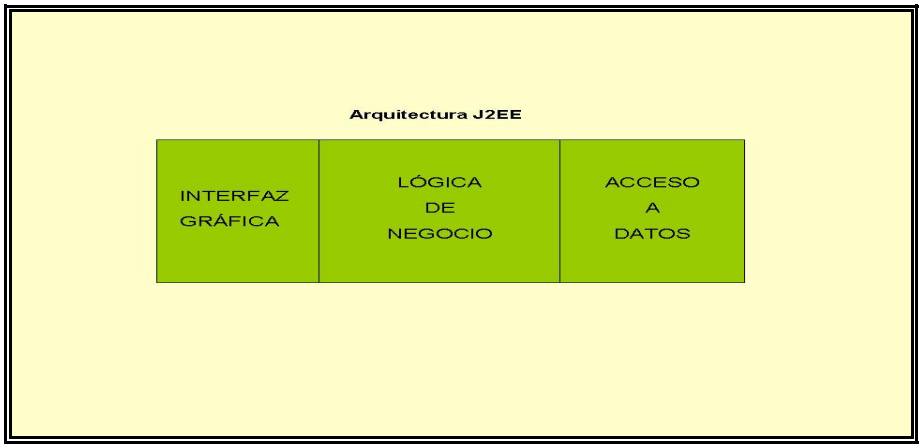

Figura 11. Arquitectura J2EE

a) Módulo de Registro

La primera capa de este módulo es la interfaz gráfica, a través de la cual se comunica con los *Services Repository People*. Para ello, cuenta con una serie de páginas Web controladas por un módulo especifico, a través de las cuales se registran, modifican Web Services, se validan usuarios, etc. La segunda capa es la de Lógica de Negocio, que consta de distintas clases Java encargadas de llevar a cabo las distintas funciones, como modificar un servicio, validar un usuario (SRP), etc. La última capa, la de acceso a datos, a través de la cual se accede al repositorio UDDI, utilizando para ello objetos DAO (Data Access Object).

b) Módulo de Entrega de Servicios (Delivery)

El módulo de entrega de servicios se asemeja mucho al modulo de registro, ya que se contemplan también las tres capas. Una de las diferencias es que aquí los usuarios son los móviles y la interfaz grafica se comunica con los usuarios a través de paginas WAP, a diferencia del uso de páginas Web por parte del módulo de registro. Además esta centrada en la capa de Lógica de Negocio, en la que encontramos el agente inteligente autónomo (MCD) de la plataforma, el cual controla las aplicaciones que gestionan los servicios

descritos. Y por último, destacar que la capa de acceso a datos interactúa además de con el repositorio UDDI, con la base de datos de perfil de usuario.

2.1.5. Taxonomía de Servicios Móviles

Dentro del Proveedor de Servicios donde se encuentra SIOS-m existen dos servicios de red básicos (simples): un servicio de personalización "Perfiles", que almacena distintas características acerca de los usuarios que han permitido que sus datos sean visibles y otro servicio que nos facilite un listado de los establecimientos que solicite el usuario. Estos dos servicios se combinarán para formar un servicio compuesto que consistirá en obtener un listado de establecimientos personalizados según el perfil elegido o de las características que el usuario ha ido indicando al sistema.

Necesitaremos un servicio exclusivo ofrecido por otro proveedor de servicios: un "localizador – callejero", que nos proporcione el itinerario necesario para llegar al destino indicado.

A continuación detallamos los diferentes servicios:

a) Servicio Básico de Personalización "Perfiles": este servicio será el encargado de almacenar distinta información acerca de todos los usuarios que hayan querido introducir sus datos personales. Estos datos pueden ser el nombre, edad, aficiones, lugar de residencia, etc. y serán utilizados por este servicio con el fin de que le puedan ser ofrecidos al usuario el resto de servicios de una manera personalizada.

Dentro de este servicio podemos encontrar diferentes opciones:

- "Mi perfil": contendrá los datos del usuario de la sesión.
- "Modificar perfil": mediante esta opción el usuario podrá crear su perfil o modificarlo. También podrá activar o desactivar la visibilidad de sus datos al resto de usuarios.
- "Buscar Perfil": podremos buscar el perfil de otra persona si ésta ha dado su permiso para que sean mostrados sus datos. El usuario podrá añadir diferentes perfiles a su lista de Perfiles Amigos. Suponemos que cada usuario de este servicio tiene un identificador único que le diferencia del resto dentro de este servicio.
- "Perfiles amigos": contendrá un "índice de amigos" para acceder de forma rápida a aquellos usuarios que el usuario haya añadido como amigos a esta lista cuando buscó un perfil.

b) Servicio Básico de Listado de Establecimientos "Páginas amarillas": este servicio devolverá al usuario un listado con todos los establecimientos del tipo que haya solicitado. Toda la información contenida en esta base de datos estará ordenada de forma geográfica.

c) Servicio Compuesto de Establecimientos Personalizados: gracias a la composición de los anteriores servicios básicos obtenemos uno nuevo, que será capaz de proporcionar al usuario un listado de establecimientos filtrado según ciertas características recogidas de los datos personales del usuario introducidos en "Perfiles". Es decir, mostrará los establecimientos que le interesen al usuario.

Así por ejemplo, si un usuario desea conocer los restaurantes de comida asiática cercanos a su domicilio, este nuevo servicio será capaz de proporcionárselos. Por un lado recogerá la lista de restaurantes asiáticos del *Servicio básico de Listado de Establecimientos "Páginas amarillas"* y por otro obtendrá la dirección del usuario accediendo a su perfil almacenado en el *Servicio básico de Personalización "Perfiles"*.

¿Y si no sabemos cómo llegar a un establecimiento desde una determinado lugar? Necesitamos por lo tanto, otro servicio exclusivo ofrecido por otro proveedor de servicios que me proporcione esta información.

d) Servicio exclusivo de Callejero – Localización: este servicio, que recordemos que se encuentra en un proveedor de servicios externo, nos proporcionará las indicaciones necesarias para llegar a un destino indicado. Este destino podrá ser un destino cualquiera o la dirección de un establecimiento previamente consultado. Este servicio pedirá al usuario si el punto origen de la ruta a planificar va a ser su domicilio (información que podrá recoger de su perfil), su posición actual, localizando la posición del terminal, previo consentimiento del usuario, mediante un sistema GPS o cualquier otra dirección que indique el propio usuario.

Facturación

Todos los servicios ofrecidos por el propio proveedor de servicios donde se encuentra SIOS-m o por un proveedor externo, deben tener una tarifa asociada al uso de dichos servicios. Para resolver los problemas con el reparto de la facturación de aquellos servicios que incluyen varios proveedores es necesaria una aplicación que se encargue de este control. Cada servicio descrito en el repositorio tendrá un coste asociado. La aplicación correspondiente al servicio será la encargada de devolver a nuestra plataforma inteligente tanto el servicio solicitado para entregárselo al usuario, como el coste asociado al mismo que será recogido por la aplicación encargada de la facturación que gestionará los movimientos de saldo del usuario propietario del terminal.

2.1.6. Estudio de Viabilidad de Sistemas Inteligentes

Basándonos en las técnicas anteriormente citadas (véase 2.1.2) y en las tecnologías investigadas, hemos llegado a las siguientes conclusiones:

- El Aprendizaje Automático no es viable para nuestro proyecto debido a que es una técnica adaptativa automática. El sistema aprenderá a partir de una colección de ejemplos de entrenamiento y nuestro sistema no puede ser tan rígido sino tiene que ser flexible y capaz de adaptarse a las necesidades del usuario.
- Los Sistemas Expertos no son viables para nuestro proyecto debido a que en un sistema experto cuando se cambia el dominio, el sistema no es capaz de adaptarse a las nuevas modificaciones.

Los sistemas basados en acciones, Agentes Software (Véase 2.1.1.3), son viables para nuestro proyecto, ya que añaden características que completan las definiciones anteriores.

Según el estudio de viabilidad de las técnicas de IA y de las diferentes tecnologías, viendo las características de nuestro sistema SIOS-m y teniendo en cuenta la división de módulos que tenemos, observamos que en el módulo de registro necesitamos una inteligencia básica, basada en conocimiento (KBAI). En cambio, necesitaremos dotar al módulo de entrega de una inteligencia basada en el comportamiento (BBAI). Por lo tanto necesitamos una arquitectura híbrida para modelar nuestro sistema, concretamente un sistema inteligente adaptativo, basado en el Mecanismo de selección de acciones [Maes, 89].

2.1.7. Análisis y Diseño de la Arquitectura SIOS-m y de la Plataforma de Simulación

El diseño de la Arquitectura SIOS-m se ha basado en la idea original de la arquitectura propuesta por nuestra profesora del proyecto (Arquitectura Mes, Motor de registro y entrega de servicios) ya comentada en la visión global de la arquitectura (Véase 2.1.2.1).

2.1.7.1. Análisis de Requisitos de los Módulos de Registro de Servicios de SIOS-m

Realizando un análisis de requisitos de los módulos de registro, vemos que para la ejecución y soporte de las distintas herramientas y aplicaciones de estos módulos serán necesarios los siguientes recursos:

- Requisitos hardware:

  - PC con acceso a Internet para la interacción de los SRP con este módulo para que se pueda llevar a cabo el registro y gestión de los diferentes Web Service.
  - Ordenador con un procesador con velocidad superior a 2GHz, una memoria principal superior a 512MB y, una unidad de almacenamiento superior a 80GB. Estos requerimientos, son debidos a que, los programas software que se van a utilizar, así como los gestores de bases de datos, requerirán de una gran potencia de cálculo.

- Requisitos software:

- JBuilder 9 Enterprise Edition, para el desarrollo de los módulos.
- JDK 1.4

- Servidor Tomcat 4.1 integrado dentro de JBuilder 9 Enterprise Edition.
- Servidor UDDI, para poder montar el repositorio donde se encuentran los Web Services.
- MySQL, JDataStore o Access para la base de datos de perfiles de usuarios de los SRP.

2.1.7.2. Análisis de Requisitos de los Módulos de Delivery de Servicios de SIOS-m

Realizando un análisis de requisitos de los módulos de delivery, vemos que para llevar a cabo la ejecución y soporte de las distintas herramientas y aplicaciones de estos módulos serán necesarios los siguientes recursos:

- Requisitos hardware:

- PC con acceso a Internet para que los usuarios finales interactúen con la aplicación a través de este módulo y se pueda llevar a cabo el uso de los servicios disponibles a través de los diferentes Web Service.
- Ordenador con un procesador con velocidad superior a 2GHz, una memoria principal superior a 512MB y, una unidad de almacenamiento superior a 80GB. Estos requerimientos, son debidos a que, los programas software que se van a utilizar, así como los gestores de bases de datos, requerirán de una gran potencia de cálculo.

- Requisitos software:

- JBuilder 9 Enterprise Edition, para el desarrollo de los módulos.
- JDK 1.4, integrado dentro del JBuilder 9 Enterprise Edition
- Servidor Tomcat 4.1 integrado dentro de JBuilder 9 Enterprise Edition.
- Servidor UDDI, para poder montar el repositorio donde se encuentran los Web Services.
- Servidor SOAP, para llevar a cabo las comunicaciones entre el proveedor de servicios y las aplicaciones correspondientes a los servicios que ofrece.
- MySQL, JDataStore o Access para la base de datos de perfiles de usuarios de la aplicación.

2.1.7.3. Diseño de Alto Nivel de los Módulos de Registro de Servicios de SIOS-m

Construimos el módulo de registro basándonos en una arquitectura modular J2EE, en la que disponemos de tres capas de implementación: capa de interfaz, lógica de negocio y la capa de acceso a datos. Cada una de ellas tendrá o una función específica detallada a continuación.

1. La capa de presentación: en la que se almacenan las paginas Web estáticas y dinámicas desde las cuales el SRP accederán a la aplicación para realizar las distintas funciones tanto con los servicios almacenados en el repositorio como con los distintos tipos de SRP.

2. La lógica de negocio donde se encuentran las clases del módulo de registro y los servlets de control de las páginas Web. Existen clases tanto para el control de los SRP como para el de los distintos servicios almacenados en el repositorio.

3. La capa de acceso a datos que contiene DAOs para el acceso al repositorio (DAO Servicios) y a la base de datos propia del registrador (DAO SRP).

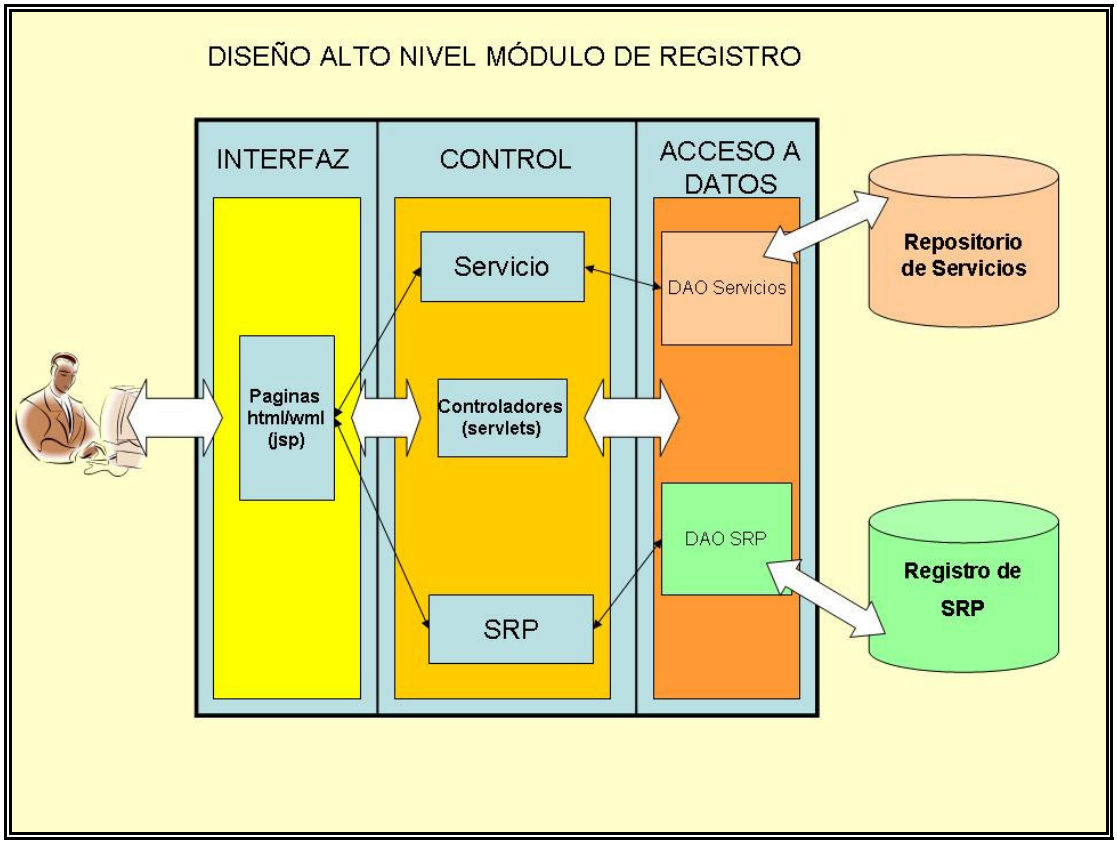

Figura 12: Diseño de Alto Nivel del módulo de registro

2.1.7.4. Diseño de Alto Nivel de los Módulos de Delivery de Servicios de SIOS-m

Basándonos en una arquitectura J2EE hemos distinguido las siguientes capas al realizar el diseño de alto nivel del módulo de Delivery:

1. Una capa de Interfaz donde se almacenan las páginas codificadas en html o wml, dependiendo del tipo de dispositivo que interactúe con el sistema. En esta capa se almacenarán las páginas específicas de Sios-m. Estas páginas pueden ser desde páginas encargadas de gestionar la parte relacionada con el registro o el login de los usuarios hasta una página de bienvenida al usuario que contendrá todos los servicios disponibles en el repositorio de servicios.

A través de esta capa pasarán también las páginas dinámicas almacenadas en cada aplicación cuando el usuario solicite algún servicio.

2. Una capa de Lógica de Negocio encargada del control del Módulo de Delivery. Aquí se encuentran una clase Servicio necesaria para tratar los servicios del repositorio y una clase Usuario necesaria para trabajar con los posibles usuarios del proveedor. Aparte de estas dos clases es necesaria otro componente que dote de inteligencia los posibles servicios y que sea capaz de formar uno nuevo de una manera inteligente. El MCD será el encargado de esta tarea mediante la clase Agente implementada, así como de controlar el redireccionamiento a las posibles páginas que se obtengan de los diferentes servicios mediante la utilización de un servlet específico. Existirán otros servlets encargados del control del alta, baja y acceso de los posibles usuarios del proveedor.

3. Una capa de Acceso a Datos donde se almacenan: un DAO Servicios encargado de acceder y obtener información del Repositorio de Servicios y un DAO Usuarios que interactúa con la base de datos donde se encuentran almacenados los usuarios del proveedor.

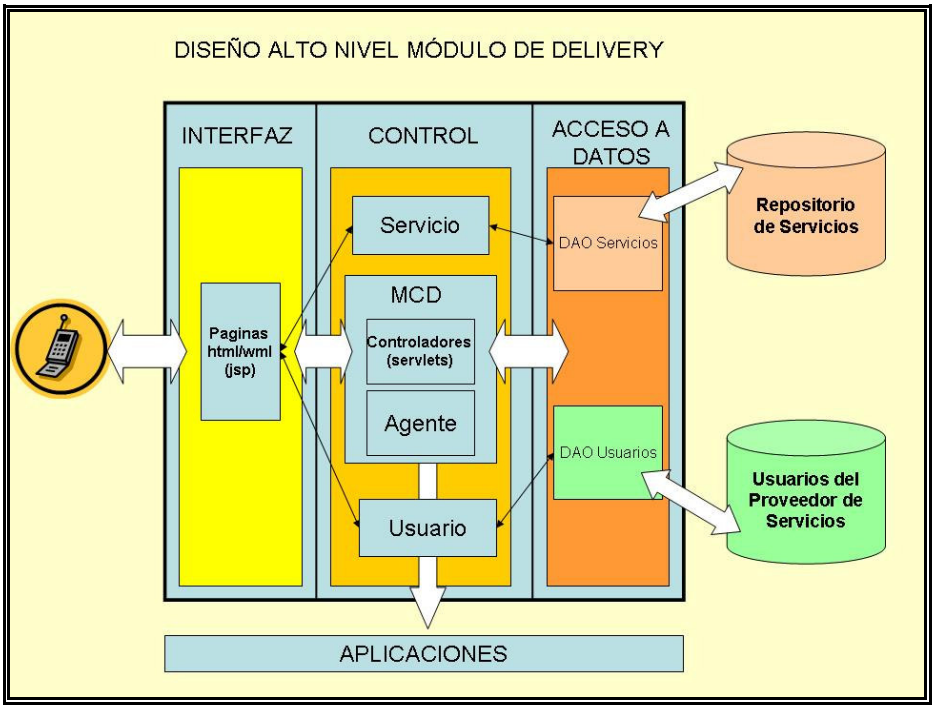

Figura 13: Diseño de Alto Nivel del modulo de delivery

2.1.7.5. Diseño Detallado de los Módulos de Registro de Servicios de SIOS-m

El diseño detallado de los módulos de registro se lleva a cabo a través de un diagrama de casos de uso del módulo de registro y su diagrama de clases correspondiente.

En el Diagrama de Casos de la figura siguiente, observamos tres actores principales que son los que hemos llamado baseRegistrador, UDDI y SRP, que a su vez, éste se descompone en tres actores específicos que son el SRP_admin, SRP_arq y el SRP_manager.

Los casos de uso contemplados en este módulo, se llevan a cabo a través de los distintos actores y son los siguientes:

- Alta_servicio: se añade un servicio nuevo al repositorio
- Busca_servicio: se busca un servicio en el repositorio.
- Alta_usuario: se registra en la base de usuarios de los SRP un usuario nuevo con el rol que le corresponda.
- Login_ususario: se lleva a cabo el acceso al módulo por parte de un SRP.
- Modifica_servicio: se modifica un servicio de los que hay en el repositorio.
- ActivarDesactivar_servicio: no todos los servicios que se encuentran en el repositorio tienen porque estar activados, por lo que se lleva un control de los servicios y los SRP son los encargados de activar o desactivar los servicios.
- Borrar_servicio: se elimina del repositorio un servicio.

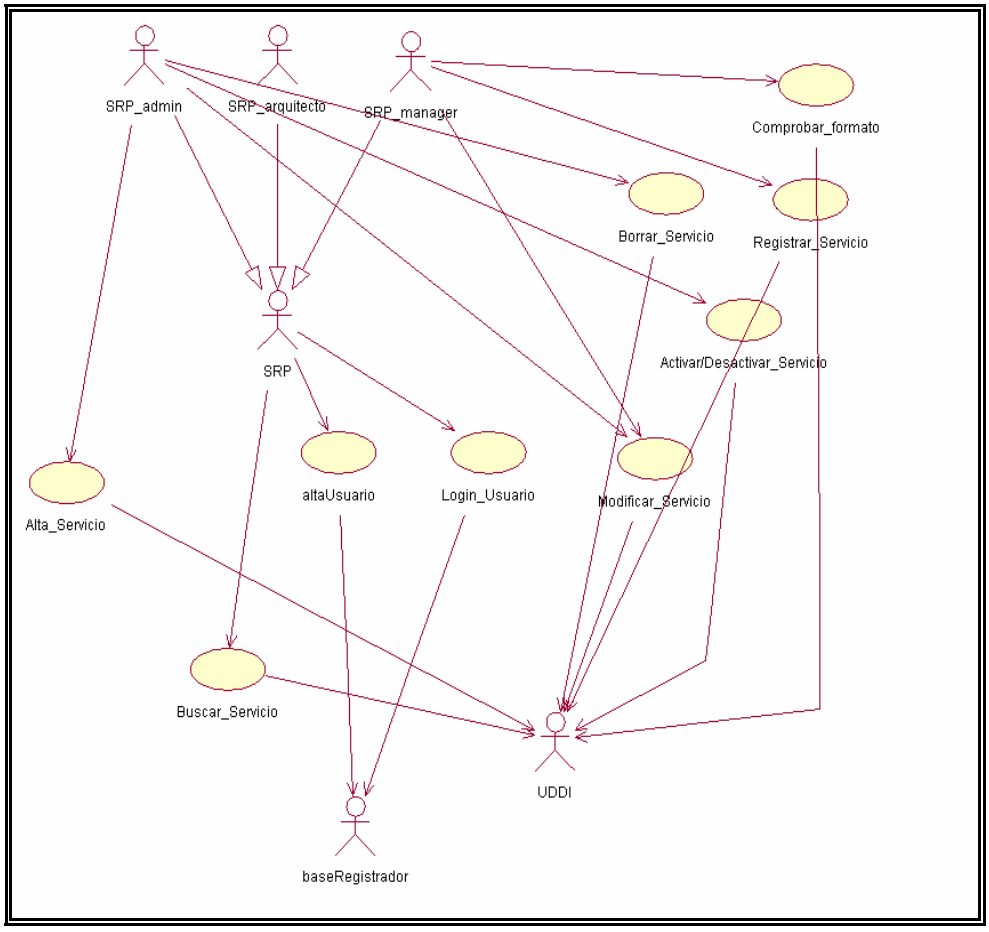

Figura 14: Diagrama de casos de uso del Módulo de Registro

En el Diagrama de clases de la figura siguiente se pueden ver tres clases principales: la clase SRP, la clase servicio y la clase registrador.

Las tres clases se corresponden, por un lado a servicio, para poder trabajar con los servicios que tenemos colocados en el repositorio, por otro lado a registrador, para que los SRP puedan registrarse dependiendo de su rol en la base de datos correspondiente y por ultimo la clase SRP, para que se gestionen los servicios del repositorio.

Hay dos DAO, uno correspondiente a la clase servicio, a través del cual se accede a la UDDI y el otro correspondiente a la clase registrador, a través de la cual se accede a la base de datos donde están registrados los SRP.

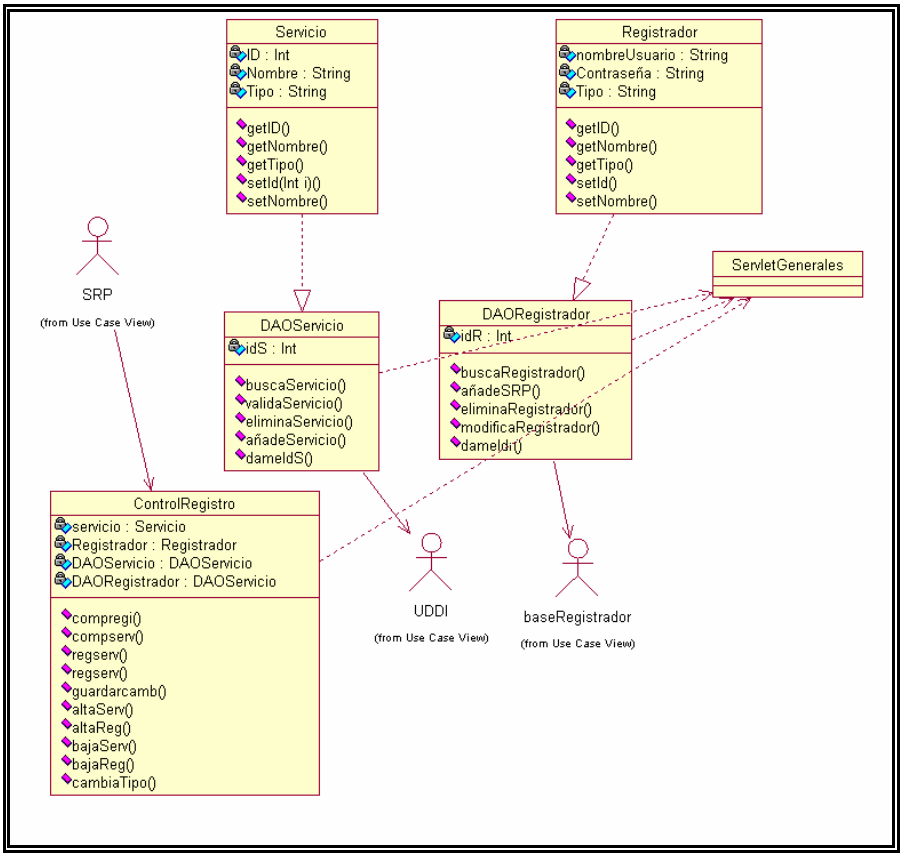

Figura 15: Diagrama de clases del Módulo de Registro

2.1.7.6. Diseño Detallado de los Módulos de Delivery de Servicios de SIOS-m

El Diseño Detallado de los módulos de Delivery se ha realizado con un diagrama de casos de usos y su diagrama de clases correspondiente.

En el Diagrama de casos de uso de la siguiente figura, se observan cinco actores principales, UsuarioPerfiles, UsuarioSios, base Usuario, baseRestaurante y AgenteInteligente. Destacamos este último, ya que es el encargado de dotar de inteligencia a este módulo, consiguiendo tener servicios simples inteligentes y un servicio compuesto, resultado de la composición de los servicios simples.

Los casos de uso contemplados en este módulo, se llevan a cabo a través de los distintos actores y son los siguientes:

- ServicioPerfiles: es el caso de uso con el que identificamos la aplicación perfiles con la que se interactúa desde el módulo de delivery.

- ServicioRestaurantes: caso de uso con el que identificamos la aplicación listado de restaurantes con la que también interactúa el módulo de delivery.

- ServicioRestaurantesPersonalizados: es el caso de uso resultante de la composición de los dos servicios simples anteriores.

- TratarFicheroConocimiento: es el caso de uso llevado a cabo por el actor principal de este módulo, es el que se encarga de interactuar con la base de conocimiento del sistema, salvando y cargando el fichero de datos.

- DAO_baseUsuario: es el caso de uso correspondiente al acceso a datos de la base de usuario.

- DAO_baseRestaurante: es el caso de uso correspondiente al acceso a datos de la base de restaurante.

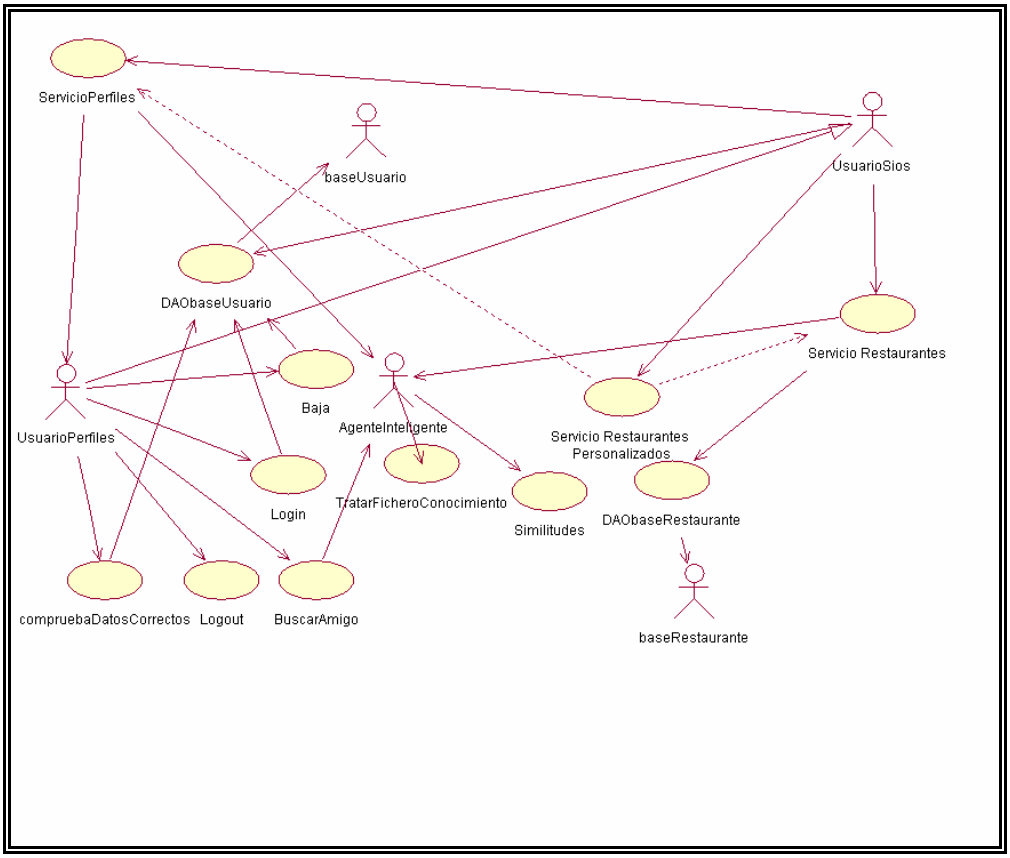

Figura 16: Diagrama de casos de uso del Módulo de Delivery

En el Diagrama de clases de la figura siguiente se pueden ver seis clases principales: ServicioDelivery, Usuario, Agente, BaseCasosUsuarios, Caso y CasosUsuario. A parte de las clases anteriormente citadas, existen todos los servlets que interconexionan las clases.

Hay cuatro DAOs, baseServicio, baseUsuarioSios, basePerfiles y baseRestaurantes. Son los encargados de acceder a los datos de las bases correspondientes a cada uno de ellos.

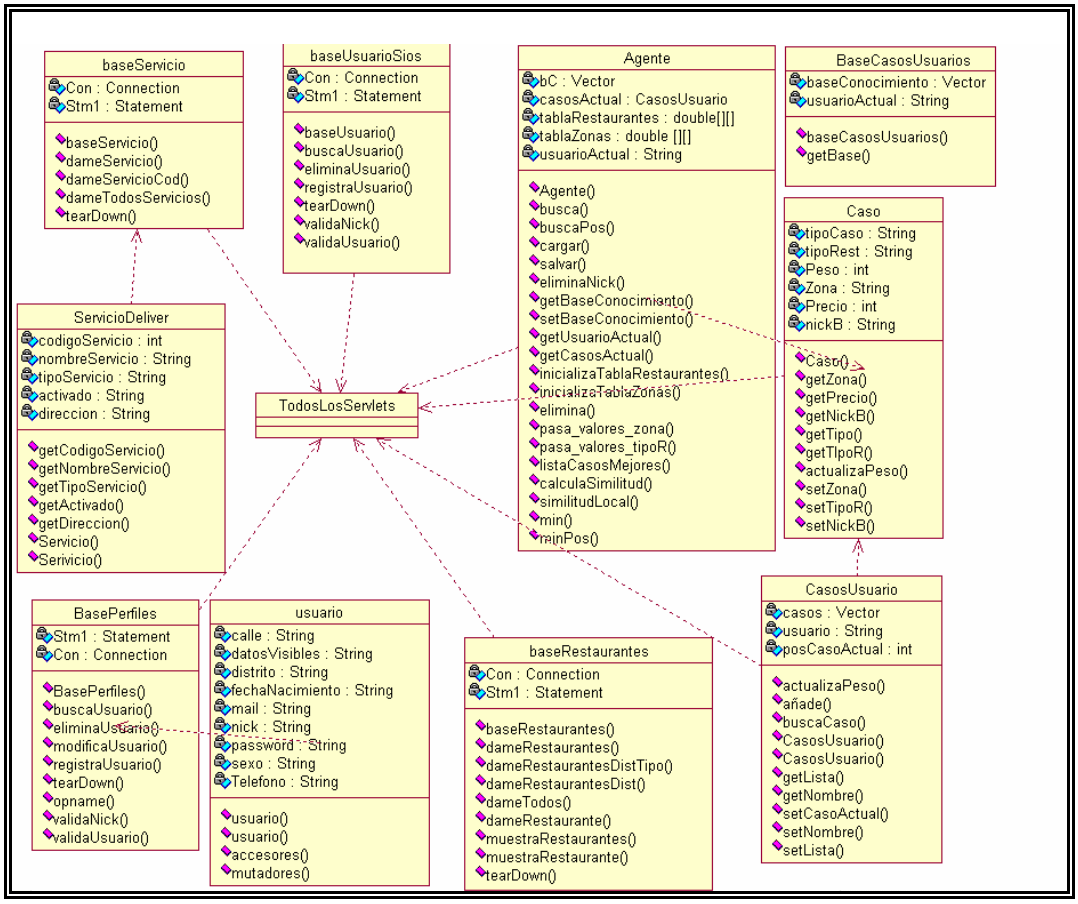

Figura 17: Diagrama de clases del Módulo de Delivery

2.1.7.7. Selección de Herramientas de Simulación

Dentro del abanico de posibilidades con el que contamos hoy en día para poder simular una arquitectura orientada a servicios, hemos realizado una selección de herramientas para llevar a cabo el diseño de la plataforma de simulación.

Estas herramientas son las siguientes: un simulador de móvil para el acceso del usuario final de la aplicación, que a través de páginas en wml se va a conectar a un servidor Tomcat y éste a través de juddi se conectará al repositorio de servicios, UDDI. El servidor Tomcat se comunicara vía SOAP con las aplicaciones de los distintos servicios que ofrece el proveedor.

2.1.7.8. Diseño de la Plataforma de Simulación

En la siguiente figura mostramos las herramientas de simulación seleccionadas.

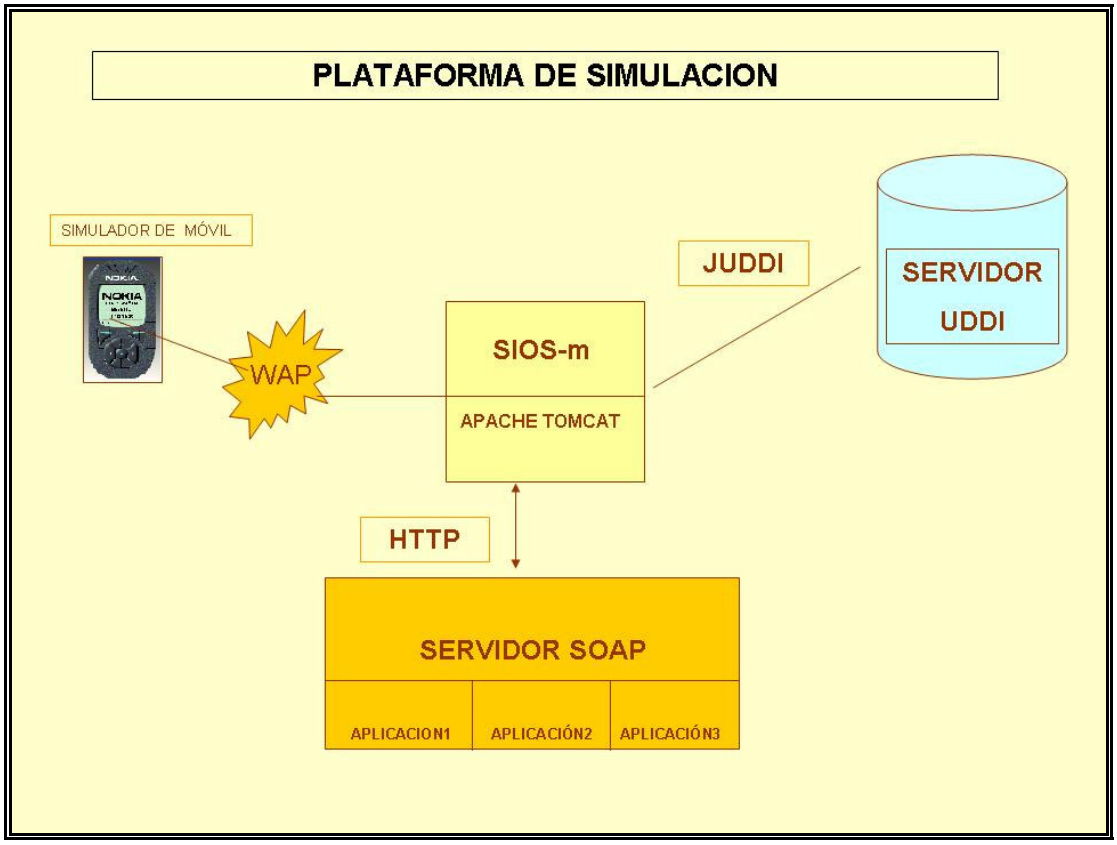

Figura 18: Plataforma de simulación**.**

2.1.8. Definición del Plan de Desarrollo y Simulación de un Prototipo SIOS-m

El plan de Desarrollo que pretendemos seguir irá en paralelo con el nivel de dificultad que pensamos que tendrán cada unos de los puntos tratados a continuación. Así, nuestro plan de desarrollo empezará con la realización de tareas más sencillas e irá avanzando en complejidad según vayamos alcanzando el objetivo deseado.

1. Desarrollaremos una aplicación en Java que sustente el servicio básico Perfiles definido en la Taxonomía de Servicios (2.1.5). Realizaremos pruebas sobre las funciones necesarias en este

servicio como serán el alta, darse de baja, ver mi perfil guardado y buscar amigos que estén dados de alta también en este servicio. Será necesaria aparte de la aplicación la creación de una base de datos donde se almacenen los datos personales de cada usuario registrado. Será necesario comprobar la correcta conexión de esta base de datos con la aplicación.

2. Una vez terminada y simulada esta aplicación implementaremos el segundo servicio básico definido: Listado de Establecimientos. Esta aplicación Java tendrá que pedir al usuario que introduzca algunas características acerca de los establecimientos que desea buscar. La información de los establecimientos tendrá que estar en una base de datos. Devolverá al usuario solicitante del servicio los establecimientos que cumplan con esas características. Será necesario comprobar la correcta conexión de la base de datos con la aplicación y que los datos devueltos se corresponden con los solicitados.

3. A continuación, y teniendo ya implementados los servicios anteriores, procederemos a la implementación de Sios-m. Como se mencionó en el diseño de alto nivel será necesario la implementación de las clases Servicio y Usuario. Estas dos clases completan su funcionalidad con la creación del repositorio de servicios y de la base de datos de usuarios del proveedor. Probaremos el correcto registro, baja y login de diferentes usuarios.

4. Implementaremos una clase Agente que será la encargada de dotar de inteligencia a las aplicaciones y de crear la aplicación compuesta establecimientos personalizados. Tendrá que ser capaz de guardar y tratar las acciones realizadas por los usuarios. Probaremos que con esta clase y gracias a las implementaciones de los servicios básicos el sistema es capaz de crear una nueva aplicación compuesta.

5. El siguiente paso será intentar simular el acceso a un servicio exclusivo que nos proporciona la información necesaria para disponer de un servicio callejero que sea capaz de orientar a nuestros usuarios.

6. El último paso será la correcta adaptación de todo lo implementado a las tecnologías estudiadas en la primera fase (UDDI, SOAP, WSDL, WML).

2.2. Fase II: Desarrollo y Simulación del Sistema SIOS-m

2.2.1. Análisis, Diseño e Implementación del Prototipo SIOS-m

Según lo detallado en la definición del plan de desarrollo (véase 2.1.8), hemos tomado las siguientes decisiones:

- Aplicación perfiles: Acceso a mi perfil, modificación, darse de baja y la posibilidad de buscar un amigo que este dado de alta en el servicio perfiles. Además de esto el sistema es capaz de aprender y recomendar el amigo con mayor peso en la base de casos de nuestro usuario.

- Aplicación restaurantes: Busca restaurantes asociados a características específicas introducidas. Para ello, realiza dos tipos de búsqueda: por un lado la búsqueda exacta, en el caso de que existan casos asociados a esas características; o bien la búsqueda aproximada, la cual devuelve los tres restaurantes mas similares en zona, tipo y precio al caso consultado.

- Aplicación restaurantes personalizados: Busca restaurantes recorriendo un árbol de decisión que el sistema crea según una serie de características introducidas en mi perfil y las búsquedas realizadas en el servicio restaurantes.

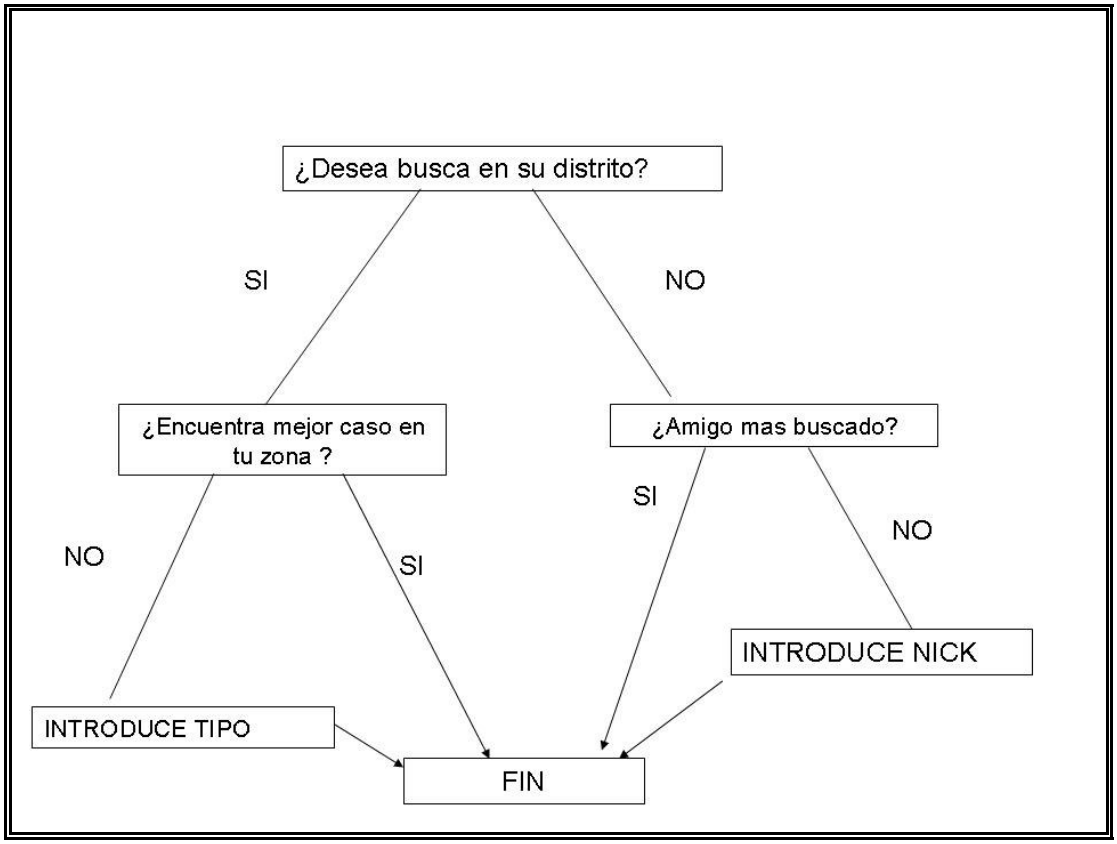

Figura 19: Diagrama de Secuencia del Servicio Personalizado

Entendiendo como fin en este árbol de decisión la obtención de los restaurantes especializados dependiendo de la rama del árbol

.El diseño del prototipo consta de un paquete principal (COM) del que cuelgan dos módulos encargados de funciones específicas. Por un lado, tenemos el paquete SIOS que centraliza el control del prototipo y, por otro está SP que contiene las aplicaciones propias de cada servicio.

1. SIOS: Paquete encargado de controlar la aplicación. A su vez consta de varios paquetes específicos: baseServicio, baseUsuario y control.

1.1 baseServicio: Paquete que comunica la tabla de la base de datos de los Servicios con el prototipo. Éste, contiene el DAO (Data Access Object) que accede a los datos de la tabla "repositorio" llamado "***baseServicio.java".***

1.1.1 "***baseServicio.java":*** Clase que hereda de "com.borland.jbuilder.unittest.JdbcFixture", que contiene los siguientes métodos:

- **baseServicio()** : constructor que inicializa los atributos heredados de jdbcfixture: url, driver, nombre de usuario y contraseña. Y, además crea las conexiones pertinentes con la base de datos.

- **void tearDown():** método que cierra las conexiones con la base de datos.

- **Servicio dameServicio(String nombreS):** método que obtiene el servicio de la tabla de la base de datos identificado por "nombreS" que esté activado.

- **Servicio dameServicio(int codS):** método que obtiene el servicio de la tabla de la base de datos identificado por "codS", únicamente si el servicio esta activado.

- **Vector dameTodos():** función que calcula todos los servicios activados de la base de datos.

1.1.2 Estructura de "Repositorio.jds"

REPOSITORIO

	CODIGOSERVICIO	ACTIVADO	DIRECCION	TIPOSERVICIO	NOMBRESERVICIO
1	0	SI	/inicioPerfiles.jsp	SIMPLE	PERFILES
2	1	SI	/inicioRestaurantes.jsp	SIMPLE	RESTAURANTES
3	2	SI	/inicioRestaurantesPersonalizados.jsp	COMPUESTO	RESTAURANTES PERSONALIZADOS
4	3	NO	/inicioCallejero.jsp	EXCLUSIVO	CALLEJERO

Figura 20 : "Respositorio.jds"

La tabla "REPOSITORIO" esta formada por los campos siguiente: codigoservicio, activado, direccion, tiposervicio y nombreservicio. Entre los cuales es el codigoservicio quien identifica a cada tupla de la tabla, por tanto la hemos establecido como clave principal.

Como podemos ver en la figura el campo ACTIVADO identifica que servicios son visibles al usuario final. En nuestro caso el servicio exclusivo callejero no será visible como una opción del menú ya que el campo ACTIVADO tiene el valor NO, en cambio los otros servicios que tienen como valor SI en dicho campo, aparecerán como opciones disponibles del menú.

1.2 baseUsuario: paquete que comunica el prototipo con la tabla de base de datos propia de los usuarios de Sios. Éste, contiene el DAO que accede a los datos de la tabla "Usuarios" llamado "***baseUsuario.java".***

1.2.1 "baseUsuario.java": Clase que hereda de "com.borland.jbuilder.unittest.JdbcFixture", que contiene los siguientes métodos:

- **baseUsuario()** : constructor que inicializa los atributos heredados de jdbcfixture: url, driver, nombre de usuario y contraseña. Y, además crea las conexiones pertinentes con la base de datos.

- **void tearDown():** método que cierra las conexiones con la base de datos.

- **boolean registraUsuario(UsuarioSios usu):** Método que se encarga de dar de alta a un usuario con su nick y contraseña comprobando que el nick introducido es válido, es decir no este ya en la tabla de la base de datos. Para ello utiliza una función boolean validaNick(String nick), la cual comprueba si ese nick que le introduces como parámetro no esta registrado en la tabla de Usuario. Este método devuelve cierto en el caso de que el registro se haya realizado con éxito.

- **boolean validaUsuario(String nick, String password):** busca en la tabla Usuario si existe uno con ese nick y contraseña asociados y devuelve cierto en ese caso.

- **UsuarioSios buscaUsuario(String nick):** Método que dado un nick obtiene el usuarioSios correspondiente en la tabla de la base de datos.

- **void eliminaUsuario(String nick):** Método que elimina un usuario de la base de datos buscándolo previamente en la tabla.

1.2.2 Estructura de "Usuario.jds"

	NICK	CONTRASEÑA
1	SUSI	sssss
2	dani	ddddd
3	PILI	ppppp
4	ANGEL	aaaaa
5	PEPI	ppppp
6	lucia	lllll
7	pablo	ppppp
8	raul	rrrrr
9	paloma	ppppp
10	lucas	lllll

Figura 21: "Usuario.jds"

Como podemos ver en la figura anterior la estructura de la base de UsuarioSios es muy simple ya que únicamente guardamos por parte del usuario referencia a su nick y su contraseña, siendo en este caso únicamente el nick el que identifique como clave primaria a un usuarioSios.

1.2 Control: Este paquete mantiene lo referente a la parte inteligente y todas las clases propias de SIOS. Incluidos en este paquete están los paquetes propios de estas clases de SIOS

1.2.1 usuario: contiene la clase ***"Usuario.java"*** que controla todo lo referente a un usuario de Sios, por tanto únicamente mantiene constructores de la clase para inicializar sus atributos: nick y contraseña, y métodos accesotes y mutadores para los mismos.

1.2.2 servicio: contiene la clase ***"Servicio.java"*** que controla todo lo referente a un servicio, llevando un control de los atributos del servicio.

1.2.3 AgenteInteligente: centraliza la parte inteligente del prototipo y, en él se incluyen paquetes como son: agentes y casos.

1.2.3.1 Agentes: controla todo lo referente a la base de conocimiento de nuestro sistema, para ello incluye una clase llamada ***"Agente.java"*** que lleva un nombre de usuario y la base de conocimiento global de nuestra aplicación, es decir aquella que guarda todos los casos de todos los usuarios, siendo el nombre de usuario el dueño de la sesión. La base de conocimiento de nuestro prototipo es un fichero de datos que cada

vez que se inicia una sesión nueva en sios se comprueba si esta vacío o no, para partir de una base de conocimiento inicialmente vacía o no. Además cada agente guarda en memoria matrices para calcular la proximidad entre zonas y tipos de restaurantes.

Esta clase (Agente.java) contiene los siguientes métodos:

- **Agente():** constructor que realiza la carga del fichero **"baseConocimiento.out",** actualizando con el valor del fichero el atributo del agente baseConocimiento. Si después de esta acción el atributo fuera vació rellenaríamos la base de conocimiento incluyendo al usuario de la sesión actual; en cambio si no fuera vació, pero el usuario no hubiera accedido añadiría una nueva posición a la base de conocimiento global con el usuario de la sesión. Posee métodos para inicializar las tablas de proximidad.

- **Vector calculaSimilitud(Vector casosNickActual, Caso c):** Este método calcula los tres casos más similares a uno dado (c) dentro de un grupo de casos (casosNickActual). Para ello lleva una vector de similitudes calculando la similitud local del caso c con respecto a cada caso de restaurantes de casosNickActual e introduciéndolo en dicho vector. De entre todas las posiciones del vector similitudes coge las tres posiciones cuyo valor sea más alto con respecto a los demás, esto lo hace con el método listaCasosMejores() .

- **void similitudLocal(Vector listaC, Caso c):** este método rellenará el vector similitudes, para ello lo primero que hace es ver si el caso c tiene relleno los campos por los que va a calcular su similitud, y añadirá el valor de la similitud calculado estáticamente accediendo a las posiciones concretas de la matriz. Para que este valor de similitud local este entre 0.0 y 1.0, simplemente realizamos una normalización llevando una cuenta de los campos rellenos en el caso.

- **double min(double a, double b, double c):** calcula el mínimo entre tres valores de doble precisión.

- **double minpos(double a, double b, double c):** calcula la posición del mínimo entre tres valores de doble precisión, siendo 1 si es a, 2 si es b y 3 si es c.

- **int pasa_valores_tipoR(String m) e int pasa_valores_zona(String m):** son para pasar al mismo valor que las constantes los valores introducidos, es decir si el String introducido es "CHINO", el método te devolverá 1 y, así accederemos correctamente a la posición de la matriz tablaTipo.

- **Accesores y mutadores** para los atributos propios de la clase.

- **CasosUsuario busca(String nombre):** busca los casos de usuario propios del usuario identificado por nombre dentro de la base de conocimiento.

- **int buscaPos(String nombre)** : obtiene la posición dentro del array de casos de usuarios para el usuario identificado por nombre.

- **void cargar():** hace un volcado de la información del fichero en el atributo propio de la clase agente: baseConocimiento. Para realizar esto usamos ObjectInputStream que permite leer el fichero como si fuera un objeto.

- **void salvar():** guarda el contenido de la base de conocimiento de la sesión en el fichero, para realizar esto usamos ObjectOuputStream y, así escribimos en el fichero un objeto de tipo vector que mantiene el contenido de la base de conocimiento.

- **void eliminaNick(String nickE):** Elimina del fichero las apariciones de nickE, tanto al usuario junto con todos sus casos y además se eliminan todas las apariciones del mismo en el resto de las listas de casos de los otros usuarios incluidos, usando para ello la función CasosUsuario elimina(CasosUsuario casosUsuario, String nickE).

1.2.3.2 casos: paquete que almacena todo lo referente a conocimiento en nuestro sistema. Consta de tres clases: BaseCasosUsuario, CasosUsuario y Caso

1.2.3.2.1 *"BaseCasosUsuario.java"*: clase que centraliza a los usuarios y sus casos, guardando en una estructura de clase todo lo que lleva el fichero con el que trabaja el agente. Consta de los siguiente métodos:

- **BaseCasosUsuarios(String user)** : constructor que inicializa los atributos propios de la clase: BaseConocimiento y usuario Actual.

- **Accesores y mutadores** para facilitar la encapsulación de los atributos dentro de la clase.

1.2.3.2.2 *"CasosUsuario.java"*: clase que guarda la información propia de un único usuario, entendiendo como tal su nombre, la lista de sus casos y, además guardamos la posición del vector de casos con mayor peso. Consta de los siguiente métodos:

- **CasosUsuario():** constructor que inicializa a por defecto los atributos propios de la clase.

- **CasosUsuario(String n) :** constructor que inicializa los atributos de un usuario determinado "n".

- **Accesores y mutadores** para los atributos de la clase.

- **boolean añade(Caso c)** : método que añade un caso a la lista de casos. Primeramente para ello comprueba si dicho caso esta ya en el vector usando el método int buscaCaso(String usuario, Caso c, Vector lc) y, si es así, no añade uno nuevo sino actualiza el peso del caso existente en una cantidad fija (0.1), pero si el caso no estuviera lo añade a la posición final del vector de casos del usuario.

- **int buscaCaso(String usuario,Caso c, Vector lc**): busca la posición de un caso en un vector de casos y, si este no existiera devolverá -1. Para este búsqueda diferenciamos si el caso buscado es de "perfiles" o "restaurantes".

- **Caso calculaMejorCaso(String tipo):** método que calcula el mejor caso en el vector de casos, entendiendo como mejor caso el de mayor peso. Para obtener el caso mejor diferenciamos caso mejor para el tipo "perfiles" o "restaurantes". Si el caso elegido no fuera lo suficientemente bueno, es decir no superara un umbral predeterminado el mejor lo elegiríamos aleatoriamente.

- **Caso calculaMejorCasoZona(String tipo,String zona):** método que calcula el mejor caso en el vector de casos en una zona determinada, es decir, escoge de entre todos únicamente los de su zona y de esos coge el de mayor peso.

- **void actualizaPeso(int gol)** : método que simplemente actualiza el peso de un caso.

1.2.3.2.3 ***"Caso.java":*** Sirve para el control de los casos de mi base de conocimiento. Realmente existen dos tipos de casos, el de "perfiles" y el de "restaurantes", pero ambos son guardados en esta estructura común. En el caso de los perfiles únicamente guardamos información acerca del último nick buscado y el peso. Mientras que para los restaurantes guardamos el tipo de restaurante, la zona y el precio. Para ambos casos el resto de los atributos permanecerá vació. Esta clase contiene los siguientes métodos:

- **Accesores y mutadores** para todos los atributos de la clase: tipoCaso, tipoR, nickB, zona, precio y peso.

- **void actualizaPeso(double gol):** método que, simplemente actualiza el peso de una caso en peso=peso +gol.

A continuación mostramos la estructura de un caso, con ejemplos particulares de cada uno:

CASO					
tipo	**nickB**	**tipoR**	**zona**	**precio**	**Peso**

Figura 22: Estructura Caso

Caso perfiles					
PERFILES	**DANI**				**0.2**
Caso Restaurantes					
RESTAURANTES		**CHINO**	**CHAMARTIN**	**50**	**0.4**

Figura 23: Ejemplo de Casos

Como podemos ver en los ejemplos los casos de perfiles tienen los campos particulares de restaurantes vacíos, mientras que los casos restaurantes tienen los campos específicos de perfiles vacíos.

1.2.4 servicios: paquete que contiene la clase ***"Servicio.java"*** para controlar los servicios dentro de nuestro prototipo en el módulo de delivery. Dicha clase contiene los atributos propios de un servicio, como son: codigoServicio, tipoServicio, nombreServicio, direccion y activado. Además de esto, contiene accesores y mutadores para cada uno de los atributos.

1.2.5 servlet: este paquete contiene los servlet exclusivos de SIOS, por tanto son los encargados de controlar las paginas jsp propias, tales como:

- **servbajasios:** realiza las comprobaciones necesarias para dar de baja a un usuario de sios y, además se vale del agente para que este elimine las apariciones de dicho usuario en la base de conocimiento global.

- **servinisios:** Comprueba si el usuario que esta intentando acceder a nuestra aplicación esta registrado o no y, además crea el agente que controlará toda la parte inteligente. Este agente como ya hemos dicho en citadas ocasiones abrirá la base de conocimiento guardada para los usuarios con anterioridad.

- **servletcomprueba:** únicamente comprueba que el usuario introducido esta en la tabla de la base de datos correspondiente.

- **servletuddi:** Este servlet realiza accesos a la tabla "repositorio" para redireccionar a la aplicación correspondiente únicamente si el servicio esta activado.

- **servreg**: Comprueba que el usuarioSios al registrarse de nuevas rellene todos los campos en el formulario de alta y registre al usuario en la base de datos.

- **servsalsios:** únicamente invalida la sesión actual.

1.2.6 usuario: paquete que contiene la clase ***"UsuarioSios.java"*** para controlar los usuariosSios dentro de nuestro prototipo en el módulo de delivery. Esta clase tiene como atributos únicamente el nick y la contraseña del usuario. Para ellos, hemos realizado accesores y mutadores.

2. SP: paquete del cual cuelgan las aplicaciones de nuestro prototipo. A su vez consta de los siguientes paquetes específicos: aperfiles, arest y apersrest.

2.1 aperfiles: paquete que contiene la aplicación para el control de las características más particulares del usuario Sios, como son el nombre completo, año nacimiento, distrito, etc.

2.1.1 basePerfiles: Paquete que comunica la tabla de la base de datos de los perfiles de usuario con la aplicación perfiles. Éste, contiene el DAO que proporciona el acceso a los datos de la tabla "baseperfiles" llamado ***"BasePerfiles.java".***

2.1.1.1 ***"BasePerfiles.java":*** Clase que hereda de "com.borland.jbuilder.unittest.JdbcFixture", que contiene los siguientes métodos:

- **BasePerfiles():** constructor que inicializa los atributos heredados de jdbcfixture: url, driver, nombre de usuario y contraseña. Y, además crea las conexiones pertinentes con la base de datos.

- **void tearDown():** método que cierra las conexiones con la base de datos.

- **boolean registraUsuario(usuario usu**): método que registra a un usuario dado en la tabla de la base de datos.

- **usuario buscaUsuario(String nick**): método que obtiene el usuario de la tabla identificado por el nick único.

- **boolean eliminaUsuario(String nick):** función que elimina al usuario de la tabla identificado por nick.

- **boolean validaUsuario(String nick, String password)** : método que comprueba la existencia de un usuario en la tabla de la base de datos identificado por nick y contraseña.

- **boolean modificaUsuario(String nick, usuario usu):** método que modifica los datos de un usuario de la tabla identificado por nick.

- **boolean validaNick(String nick):** método que valida sólo el nick del usuario, para comprobar que no existe otro igual.

2.1.1.2 Estructura de "BasePerfiles.jds"

	TELEFONO	SEXO	PASSWORD	NOMBRE_COMPLE...	NICK	MAIL	FECHA_NACIMIEN...	DISTRITO	DATOS_VISIBL...	CALLE
1	ppp	F	ppppp	Elena Pilar	PILI	ppp	8/11/1981	ALUCHE	SI	ppppp
2	222	M	aaaaa	aaaaangel	ANGEL	aa	222	ALUCHE	SI	222
3	ll	F	llll	Lucia	lucia	ll	ll	CHAMARTIN	SI	l
4	91	M	ppppp	Paloma	paloma	pppp	154	CHAMARTIN	SI	papapa
5	ee	F	eeeee	eee	elena	ee	ee	ALUCHE	SI	ee
6	asdkflas	M	aaaaa	alan	alan	asdfj	08/11/1963	ALUCHE	SI	asdfasd
7	qq	M	ddddd	dani	dani	qq	qq	CHAMARTIN	SI	qq

Figura 24 : "BasePerfiles.jds"

La tabla "BASEPERFILES" esta formada por los campos siguientes: teléfono, sexo, password, nombre_completo, nick, mail, fecha_nacimiento, distrito, datos_visibles, calle. Entre los cuales es el nick quien identifica a cada tupla de la tabla, por tanto la hemos establecido como clave principal.

Como podemos ver en la figura el valor del campo datos visibles puede ser si o no, entendiendo como tal si al usuario le interesa o no que sus datos personales sean visibles para otros usuario o no.

2.1.2 Control: paquete que centraliza las acciones de la aplicación perfiles, es decir la funcionalidad de la aplicación. Dentro del mismo tenemos dos paquetes, uno para los servlet y otro con la clase usuarioPerfiles.

2.1.2.1 servlet: este paquete contiene los servlet encargados de controlar las páginas jsp propias de esta aplicación, entre ellos están:

- **ServletBaja**: encargado de eliminar al usuario de la aplicación perfiles y todas las apariciones del mismo en la base de conocimiento de nuestro prototipo.

- **ServletBuscar:** es el que se encarga de buscar al usuario introducido, comprobando para ello primero si esta en la base de perfiles y si sus datos son visibles. A la hora de realizar esta búsqueda el agente se encargará de guardar los casos correspondientes a perfiles, para así la siguiente vez que entres a buscar un nick, recomendaremos el caso de los guardados con mayor peso.

- **servletbuscarint:** encargado simplemente de mostrar los datos del nick aconsejado por el sistema, con el consiguiente aumento del peso del caso en particular.

- **ServletDatos**: Comprueba a la hora de rellenar los datos del formulario perfiles que todos los campos están rellenos y, lo guarda en la base de perfiles.

- **ServletInicio**: valida al usuario en la base de datos para permitirle o no acceso a el servicio perfiles.

- **servletSalir**: Guarda en el fichero todos los casos de la base de conocimiento que he ido añadiendo durante el uso del servicio perfiles.

2.1.2.2 usuarios: paquete que contiene la clase ***"usuario.java"*** para el control centralizado de los usuarios de la aplicación perfiles, por tanto guarda como atributos aquellos que hemos guardado en la base de datos de perfiles.

2.2 arest: paquete que contiene la aplicación para el control de las características más particulares de los restaurantes, como son el nombre, teléfono, zona, etc.

2.2.1 baseRestaurantes: Paquete que comunica la tabla de la base de datos de los restaurantes con la aplicación restaurantes. Éste, contiene el DAO que proporciona el acceso a los datos de la tabla "baseRestaurantes" llamado ***"baseRestaurantes.java".***

2.2.1.1 "baseRestaurantes.java": Clase que hereda de "com.borland.jbuilder.unittest.JdbcFixture", que contiene los siguientes métodos:

- **baseRestaurantes ():** constructor que inicializa los atributos heredados de jdbcfixture: url, driver, nombre de usuario y contraseña. Y, además crea las conexiones pertinentes con la base de datos.

- void tearDown(): método que cierra las conexiones con la base de datos.

- **Vector dameRestaurantes(String Tipo, String Zona, int Preci**o): Método que obtiene un listado con los restaurantes que cumplen los argumentos pasados como parámetros.

- **restaurante dameRestaurante(String Tipo, String Zona, int Precio):** método que obtiene el restaurante que tenga como tipo, zona y precio el indicado como parámetro.

- **void muestraRestaurantes(Vector ListaRest), void muestraRestaurante(restaurante rest**): métodos que muestran por pantalla la información correspondiente a lo pasado como parámetro, bien un restaurante o una lista de ellos.

- **Vector dameRestaurantesDist(String zona)** : método que obtiene todos los restaurantes de la "zona".

- **Vector dameRestaurantesDistTipo(String zona, String tipo):** método que obtiene todos los restaurantes de una "zona" y "tipo" determinados.

- Vector dameTodos(): método que obtiene todos los restaurantes incluidos en la tabla de restaurantes**.**

2.2.1.2 Estructura de "baseRestaurantes.jds"

	NOMBRE	TIPO	ZONA	PRECIO	DIRECCION	TELEFONO
1	La gran muralla	CHINO	CHAMARTIN	40	Narcisos 66	914125689
2	Pizzoleto	ITALIANO	ALUCHE	30	Tortolosa 45	915437612
3	Don Lolailo	ESPAÑOL	CHAMARTIN	40	Cochabamba 78	914208563
4	Chinatown	CHINO	BARRIOCONCEP	40	Ferrol 13	914174537
5	La perla	CHINO	CHAMARTIN	40	Ferraz 79	914886785

Figura 25 : "baseRestaurantes.jds"

La tabla "BASERESTAURANTES" esta formada por los campos siguientes: nombrerest, tipo, zona, precio, dirección y teléfono. Entre los cuales es el nombre de restaurantes quien identifica a cada tupla de la tabla, por tanto la hemos establecido como clave principal.

Como podemos ver en la figura el valor del campo tipo puede ser para nuestro prototipo: chino, italiano y español. El campo zona puede tomar únicamente los siguientes valores: chamartin, aluche y barrioconcepcion

2.2.2 Control: paquete que centraliza las acciones de la aplicación restaurantes, es decir la funcionalidad de la aplicación. Dentro del mismo tenemos dos paquetes, uno para los servlet y otro con la clase restaurante.

2.2.2.1 servlet: este paquete contiene los servlet encargados de controlar las páginas jsp propias de esta aplicación, entre ellos están:

- **servletres:** si la búsqueda del usuario dio resultados porque existían restaurantes con dichas características, el caso se añadirá a la base de casos específica de dicho usuario. En cambio, si el sistema recomendó restaurantes, no se añaden los casos de los restaurantes recomendados a la base de conocimiento.

- **servletsalirrest**: Guarda en el fichero todos los casos de la base de conocimiento que he ido añadiendo durante el uso del servicio restaurantes.

- **servrestint**: encargado simplemente de mostrar los datos del restaurante aconsejado por el sistema, con el consiguiente aumento del peso del caso en particular.

2.2.2.2 usuarios: paquete que contiene la clase "***restaurante.java"*** para el control centralizado de los restaurantes de la aplicación restaurantes, por tanto guarda como atributos aquellos que hemos guardado en la base de datos de restaurantes.

2.3 arestpers: paquete que contiene la aplicación personalizada de búsquedas de restaurantes según características particulares de los perfiles de los usuarios que el usuario final determine, es decir busque según su perfil o el de un amigo, bien puede servir esto por si te interesa buscar los restaurantes que le gusten a un amigo concreto.

2.3.1 control.servlet: este paquete contiene los servlet encargados de controlar las páginas jsp propias de esta aplicación, entre ellos están:

- **servamirest**: personaliza la búsqueda de restaurantes según la zona donde vive tu amigo, bien puede ser el amigo que el sistema te recomiende (el de mayor peso) o el que le indiques.

- **servbuscrest**: se encarga de buscar los restaurantes de una zona determinada, incluyendo los casos en la base de conocimiento que existe por cada usuario de la aplicación.

- **servcompers**: comprueba que el usuario esté o no dado de alta en el servicio perfiles. Si no esta dado de alta le da el sistema la opción de ir al formulario a darse de alta.

- **servexacto**: comprueba que existen restaurantes almacenados en la base de datos de restaurantes de un distrito y zona concreta, en este caso introducidos por pantalla.

- **servrestpers**: Devuelve una lista de restaurantes personaliza, para ello comprueba si la lista de casos asociada al usuario de la sesión es vacía o no. En el caso de que sea vacía pregunta por el tipo de restaurante por el que quieres buscar; pero en cambio si no es vacía cojo de entre ellos el caso mejor de la zona del usuario y le recomiendo.

- **servsalvpers**: Guarda en el fichero todos los casos de la base de conocimiento que he ido añadiendo durante el uso del servicio restaurantes personalizados.

- **servtipopers**: Obtiene los restaurantes asociados a su distrito y con el tipo de restaurante obtenido de calcular el caso mejor de todos los restaurantes de su zona.

- **servzontip**: busca los restaurantes de la base de datos asociados a un tipo introducido con la zona de usuario cogida de su perfil de la base perfiles.

2.2.2. Instalación, Configuración y Prueba de la Plataforma de Simulación

- Installer JBuilder 9.0 Enterprise Edition
- Configurar el servidor de aplicaciones Tomcat para la generación, compilación y prueba de JSPs.
- Asociar el servidor Tomcat 4.1 al proyecto
- Desarrollar los JSPs de la aplicación
- Ejecutar Tomcat y probar los JSPs.
- JBuilder muestra los mensajes de ejecución del servidor. Cuando en la consola de JBuilder se ve la línea "INFO: Starting Coyote HTTP/1.1 on port 8080", se puede abrir un browser y solicitar la ejecución de los JSPs desde la siguiente página: http://localhost:8080/

2.2.3. Definición del Plan de Pruebas Funcionales y de Rendimiento del Prototipo SIOS-m

Según lo detallado en la definición del plan de desarrollo (Véase 2.1.8) y el prototipo que se ha conseguido implementar finalmente (Véase 2.2.1), las pruebas funcionales que se van a llevar a cabo son las siguientes:

- 1.Probar la aplicación que sustenta el servicio perfiles
- 2.Teniendo en cuenta las decisiones tomadas en el análisis del prototipo, probar la aplicación que sustenta el servicio listado de establecimientos, particularizándolo solamente para restaurantes.
- 3.Probar el funcionamiento de Sios-m: registro, baja y login de diferentes usuarios.
- 4.Probar la perfecta integración en Sios-m de las aplicaciones de los servicios, junto con el Agente desarrollado y el funcionamiento del servicio compuesto logrado a partir de los servicios simples y la inteligencia del Agente.

2.2.4. Instalación y Configuración del Prototipo SIOS-m

Para llevar a cabo la instalación y configuración del prototipo, hemos instalado el JDK 1.5 y el Tomcat externo 5.0.28, configurando las variables de entorno correspondientes para su correcto funcionamiento. Colgando la aplicación que hemos desarrollado en el servidor Tomcat instalado y configurado, colocando en la carpeta ROOT nuestro WEB-INF y todos nuestros JSPs, podemos probar el prototipo desarrollado.

2.2.5. Realización del Plan de Pruebas Funcionales y de Rendimiento del Prototipo SIOS-m

Gracias al haber hecho una definición de pruebas factible viendo las limitaciones con las que nos hemos encontrado se han realizado con éxito las pruebas especificadas anteriormente.

2.2.6. Análisis de Resultados y Optimización de la Arquitectura SIOS-m

En resumen, concluimos con un diseño simple de un sistema Web que es controlado mediante la acción inteligente de un único agente, cuyo conocimiento esta basado en casos. Este agente es la parte fundamental de nuestro prototipo y se encarga de mantener los casos, esto es las búsquedas del usuario, en un fichero al cerrar la sesión y, así mantiene para futuras sesiones el rastro del usuario en la aplicación.

3. Líneas de Evolución de SIOS-m

- Estudio profundizado del módulo de registro: funcionamiento, gestión, optimizaciones,... (Recordamos que nosotros suponemos ya rellenada la UDDI con la descripción de los servicios).
- Aumentar Usuarios finales: PDA como dispositivo móvil, otro proveedor de servicios.
- Aumentar el número de roles de los *Services Repository People*.
- Adaptación de la plataforma a las nuevas características que vayan apareciendo de los terminales.
- Introducir nuevos servicios (básicos, compuestos, servicios de otros proveedores de servicios,...).
- Montar un servidor UDDI local para poder alojar Web Services.
- Convertir cada aplicación de servicios en un Web Services.
- Publicar Web Services en la UDDI local.

4. Conclusiones

El campo de las tecnologías móviles abarca un extenso ámbito de posibilidades y engloba un amplio conocimiento de nuevas tecnologías, como son: soap, uddi, etc. Las cuales no han llegado a ser configuradas, pero si investigadas, debido a la complejidad de las mismas a la hora de la instalación y de la decisión tomada de centrarnos principalmente en la implementación de la parte inteligente de SIOS-m, dejando para una futura continuación del proyecto la terminación de lo original planeado.

Es un proyecto ambicioso y con la posibilidad de extenderse por muchos caminos, por tanto tomamos la decisión de acotar los límites del proyecto para que fuera factible el diseño de un prototipo básico.

5. Bibliografía

E. Newcomer, Understanding Web services: XML, WSDL, SOAP, and UDDI, Addison-Wesley,

Boston, Mass., May 2002.

F. Curbera et. al., “Unraveling the Web Services Web: An Introduction to SOAP, WSDL, and

UDDI,” IEEE internet Computing, vol. 6, no. 2, March/April 2002, pp. 86-93.

D. Chapell. “Distributed Computing Tomorrow: Web Services and Beyond”, Microsoft Tech-Ed

2002, Barcelona.

S. Vinoski, "Web Services Interaction Models — Part 1: Current Practice," IEEE internet Computing, vol. 6, no. 3, May/June 2002, pp. 89-91.

6. Links y Otras Referencias

Arquitecturas Móviles

[1] Web Services Architecture, W3C Working Group Note, D. Booth, H. Haas, F. McCabe, E. Newcomer, M. Champion, C. Ferris, D. Orchard, 11 February 2004 (http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/.)

[2] Web Services Architecture Usage Scenarios, W3C Working Group Note, H. He, H. Haas, D. Orchard, 11 February 2004 (http://www.w3.org/TR/2004/NOTE-ws-arch-scenarios-20040211/.)

[3] B. Sleeper. The five missing pieces of SOA. September 10, 2004 http://www.infoworld.com/article/04/09/10/37FEWebservmiddle_1.html

[4] http://www.w3.org

[5] http://www.w3.org/2002/ws/

[6] http://www.w3.org/TR/wsdl/

[7] http://www.uddi.org/

[8] Desarrollos SOA: http://dev2dev.bea.com/soa/

[9] Services and SOA: http://www.service-architecture.com/

[10] SOA Technologies: http://www.intelligententerprise.com/031030/617/feat2_1.html

[11] Enterprise Service Platform: http://www.Webmethods.com/

[12] SOA & Web Services: http://www.ibm.com/developerworks/Webservices/

[13] HP Middleware: http://www.hpmiddleware.com/Webservices

[14] Web Services Topics: http://Webservives.xml.com/pub/a/ws/2001/04/04/Webservices/

Servicios Móviles

[15] Telefónica I+D: www.tid.es

[16] International Network Services: http://www.ins.com/

[17] Network and WAN Services: http://www.networkworld.com/topics/wan.html

[18] Value Added Services: http://www.isp-planet.com/services/

[19] Wireless Services: http://wirelessadvisor.com/

Tecnología y estándares

[20]Java Technology: http://java.sun.com

[21]J2EE: http://java.sun.com/j2ee/index.jsp

[22]J2EE Diseño y Desarrollo: http://dev2dev.bea.com/

[23]J2EE Developers: http://j2eedevgroup.dev.java.net/

Ejemplos de Web Services

[24] Ejemplo de Web Services: www.spincircuit.com

[25] Arquitectura de Microsoft: www.microsoft.com

7. Glosario de Términos

Por orden de aparición en este documento.

Inteligencia Artificial: inteligencia exhibida por artefactos creados por humanos (es decir, artificiales).

Plataformas: sistemas complejos que a su vez sirven para crear programas.

Tecnología: conjunto ordenado de conocimientos y los correspondientes procesos que tienen como objetivo la producción de bienes y servicios, teniendo en cuenta la técnica, la ciencia y los aspectos económicos, sociales y culturales involucrados.

Agente: entidad inteligente que existe dentro de un cierto contexto y que se puede comunicar a través de un mecanismo de comunicación inter - proceso.

Estándar: que sirve como tipo, modelo, norma, patrón o referencia.

Taxonomía: ciencia que trata de los principios, métodos y fines de la clasificación.

Viabilidad: cualidad que tiene un asunto, que por sus circunstancias, tiene probabilidades de poderse llevar a cabo.

Software: conjunto de programas, instrucciones y reglas informáticas para ejecutar ciertas tareas en una computadora.

Prototipo: ejemplar original o primer molde en que se fabrica una figura u otra cosa.

Simulación: representación de algo, fingiendo o imitando lo que no es.

Planificación: plan general, metódicamente organizado y frecuentemente de gran amplitud, para obtener un objetivo determinado.

Infraestructura: conjunto de elementos o servicios que se consideran necesarios para la creación y funcionamiento de una organización cualquiera.

Web Service: aplicación software identificada mediante una URI, cuyos interfaces públicos y enlaces se definen y describen usando XML.

Repositorio: lugar central donde la información es almacenada y mantenida.

Aplicaciones: programas preparados para una utilización específica.

Java: lenguaje de programación orientado a objetos desarrollado en la empresa Sun Microsystems a principios de la década del 90.

.NET: interfaz para programar aplicaciones desarrollado por Microsoft.

Protocolos (de red): conjunto de reglas que controlan la secuencia de mensajes que ocurren durante una comunicación entre entidades que forman una red.

Interfaz: conexión física y funcional entre dos aparatos o sistemas independientes.

Internet: red informática mundial, descentralizada, formada por la conexión directa entre computadoras u ordenadores mediante un protocolo especial de comunicación.

Facsímil: perfecta imitación o reproducción de una firma, de un escrito, de un dibujo, de un impreso, etc.

Red: conjunto de elementos organizados para determinado fin.

Vocabulario: conjunto de palabras de un idioma pertenecientes al uso de una región, a una actividad determinada, a un campo semántico dado, etc.

Desarrolladores: informáticos que programan aplicaciones en distintos lenguajes de programación informáticos.

Hipertexto: documento digital o no, que se puede leer de manera no secuencial.

Navegador (Web): aplicación software que permite al usuario recuperar y visualizar documentos de hipertexto, comúnmente descritos en HTML, desde servidores Web de todo el mundo a través de Internet.

Servidor: aplicación informática o programa que realiza algunas tareas en beneficio de otras aplicaciones llamadas clientes.

Servlets: objetos que corren dentro del contexto de un servidor Web (Ej.: Tomcat) y extienden su funcionalidad.

Patrones: modelos que sirve de muestra para sacar otra cosa igual.

Enlace: algo que conecta dos objetos diferentes.

Rol: función que alguien o algo cumple.

Portal: plataforma que posee una Infraestructura propia funciona como middleware, por lo tanto es una Plataforma que integra múltiples aplicaciones (sin importar su arquitectura o plataforma) en un solo Frontend dentro de un Browser, el cual se puede acceder desde cualquier sitio, en cualquier momento y con cualquier dispositivo de forma sencilla y segura.

Sistema: sistema informático completo. Incluye software, middleware y hardware.

8. Acrónimos

Por orden de aparición en este documento.

SIOS-m: Sistema Inteligente Orientado a Servicios Móviles

IA: Inteligencia Artificial

WS: Web Services

SOA: Service Oriented Architecture

UDDI: Universal Description, Discovery, and Integration

WSDL: Web Service Description Language

SOAP: Simple Object Access Protocol

W3C: World Wide Web Consortium

URI: Uniform Resource Identifier

XML: eXtensible Markup Language

URL: Uniform Resource Locator

HTTP: Hypertext Transfer Protocol

SMS: Short Message System

RPV: Red Privada Virtual

SGML: Standard Generalizad Markup Language

J2EE: Java 2 Enterprise Edition

IBM: International Business Machines

DCOM: Distributed Component Object Model

CORBA: Common Object Request Broker Architecture

SMTP: Simple Mail Transfer Protocol

APIs: Application Programming Interfaces

NDR: Network Data Representation

CDR: Common Data Representation

ASP: Active Server Pages

JSP: Java Server Pages

EJB: Enterprise JavaBeans

PC: Personal Computer

PDA: Personal Digital Assistant

SW: Software

HW: Hardware

KBAI: Knowledge-Based Artificial Intelligence

BBAI: Behaviour-Based Artificial Intelligence

ASM: Action Selection Mechanism

ISP: Internet Service Provider

NSP: Network Service Provider

MCD: Módulo de Control del módulo Delivery

SRP: Services Repository People

BD: Base de Datos

PS: Proveedor de Servicios

DAO: Data Access Object

WAP: Wireless Application Protocol

GPS: Global Position System

9. LISTADO DE PALABRAS CLAVE

- Agente
- JSP
- UDDI
- SOAP
- Web Services
- J2EE
- Tomcat
- WAP
- Dispositivo Móvil
- SOA
- WSDL

▪

10. AUTORIZACIÓN

Mediante el texto siguiente, se autoriza a la Universidad Complutense de Madrid a difundir y utilizar, con fines académicos, no comerciales y mencionando expresamente a sus autores, tanto la propia memoria, como el código, la documentación y/o el prototipo desarrollado.

Firmado:

Susana Bautista Blasco **Daniel González Polo**

Elena Pilar Rueda Moltó